

©Copyright 2022

Tongshuang Wu

Interactive AI Model Debugging and Correction

Tongshuang Wu

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:

Jeffrey Heer, Chair

Daniel S. Weld, Chair

Marco Tulio Ribeiro

Program Authorized to Offer Degree:
Paul G. Allen School of Computer Science & Engineering

University of Washington

Abstract

Interactive AI Model Debugging and Correction

Tongshuang Wu

Co-Chairs of the Supervisory Committee:

Noe Endowed Professor Jeffrey Heer

Paul G. Allen School of Computer Science & Engineering

Thomas J. Cable / WRF Professor Daniel S. Weld

Paul G. Allen School of Computer Science & Engineering

While the accuracy of Natural Language Processing (NLP) models has been improving, users have expectations beyond what is captured by standard performance metrics. For example, chatbot assistants should not provide inappropriate or unfair responses to certain types of inquiries, and translators should not sacrifice their support for languages with low resources for perfect performances in English. Unfortunately, existing models have various deficiencies (*e.g.*, too sensitive to trivial input perturbations), thereby creating a gulf between “accurate models” (those that place high on leaderboards) and “successful models” (those that can support real world use cases).

In this work, we argue that this gap exists primarily because we are not considering human needs sufficiently throughout the model development cycle. We present how the *human perspective* is missing across the model development and deployment, and address the issues by building tools to interactively help humans debug and correct models. First, the *evaluation* that developers conduct (*e.g.*, holdout accuracy) does not reflect human expectations on how models should behave (*e.g.*, models should be right for the right reasons). For experts to express their expectations regarding models, we provide them with domain-specific languages that allow grouping similar examples and performing counterfactual, what-if analyses, so that they can rigorously inspect the model on a variety of concrete capabilities. Second, the *data* that model developers collect for building an NLP

model usually contains biases and distribution gaps, and does not reflect how humans will actually use the model. To compensate for human omissions in defining, collecting, and inspecting the intended data distribution, we build automated approaches (NLP text generators, automatic pattern mining and sampling algorithms) that can augment experts in collecting how humans use models. Third, the *default interactions* with deployed models do not allow *end users* to recover from AI errors. To make AIs more usable in downstream applications, we also design interaction strategies that help end users collaborate with deployed AIs in a transparent and controllable manner so they can detect and overwrite AI errors in real-time.

Taken together, this thesis shows that when given strategies and tools to interactively massage (partition, perturb, and decompose) data throughout the machine learning model development stage, developers and end users can debug and correct AI models in a more comprehensive, less biased, transparent, and controllable way.

TABLE OF CONTENTS

	Page
List of Figures	iv
Chapter 1: Introduction	1
1.1 Thesis Overview	2
1.2 Thesis Statement	8
1.3 Prior Publications and Authorship	9
Chapter 2: Helping Experts Express Domain Expertise for Scalable and Reproducible Error Analysis	10
2.1 Introduction	10
2.2 Task, Dataset, and Model	13
2.3 Error Analysis Principles & Errudite	14
2.4 Interactive User Interface	19
2.5 User Study	21
2.6 Related Work	25
2.7 Conclusion and Discussion	26
Chapter 3: Augmenting Experts via Systematic Grouping for Assessing Dataset Assessment	28
3.1 Introduction	28
3.2 Understanding Query Analysis And Grouping	31
3.3 The Concept of Structural Template	33
3.4 The Tempura Interface	37
3.5 Scoring Traversals	39
3.6 Template Summarization	41
3.7 Usage Scenario	44
3.8 User Study	47
3.9 Related Work	51

3.10	Discussions	53
Chapter 4:	Augmenting Experts via Counterfactual Reasoning for Explaining, Evaluating, and Improving Models	56
4.1	Introduction	56
4.2	General-Purpose Counterfactuals	58
4.3	Counterfactual Evaluation & Training	63
4.4	Counterfactual Explanations	67
4.5	Interactive Analysis	70
4.6	Related Work	72
4.7	Conclusion and Future Work	73
Chapter 5:	Augmenting Experts via Few-Shot Example Selection for Effective In-context Learning	75
5.1	Introduction	75
5.2	ScatterShot: Design and Implementation	78
5.3	Case Studies	84
5.4	User Study Planning	88
5.5	Related Work	89
5.6	Discussion	91
Chapter 6:	End Users Debugging LLMs: Transparent and Controllable Human-LLM Interaction through AI Chaining	92
6.1	Introduction	92
6.2	Background and Related Work	96
6.3	Chaining LLMs	100
6.4	Interactive User Interface	106
6.5	User Study	109
6.6	Case Studies	122
6.7	Discussion & Future Directions	125
6.8	Conclusion	128
Chapter 7:	Conclusion and Future Work	129
7.1	Discussion: The Importance of Data	130

7.2 Opportunities for Future Research	131
Appendix A: Additional Details for Errudite	177
A.1 Additional Use Cases	177
A.2 Programming-by-Demonstration	181
A.3 Survey: Error Analysis Sample Sizes	184
A.4 DSL Documentation	185
Appendix B: Additional Details for Polyjuice	188
B.1 GPT-2 as Counterfactual Generator	188
B.2 Additional Train & Eval Details, §4.3	191
B.3 Additional Explanation Details §4.4	194
B.4 Additional Err. Analysis Details §4.5	195
Appendix C: Additional Details for AI Chains	198
C.1 Identifying LLM Primitive Operations	198
C.2 Additional Details for User Study	199
C.3 Full LLM Chains for Case Studies	205
C.4 The Full Implementation of Primitive Operations	206

LIST OF FIGURES

Figure Number	Page
1.1 Thesis overview: We present how the human perspective is missing in three major model development stages, and address the issues by building build tools to interactively help humans debug and correct models.	2
1.2 Errudite example. This tool enables scalable and testable error analysis through <i>systematic grouping</i> and <i>counterfactual rewriting</i> . (A) A visual question answering model predicts “How many people...” correctly but “How many brownish...” incorrectly. Experts suspect the adjectives make a difference. (B) They scale up the observation , by building groups of questions that contain \$ADJectives, or start with \$“How many ADJ”. (C) They also test the root error cause with rewrite rules that answer: “ <i>If the adjectives were not there, would the model predict correctly?</i> ” . . .	4
1.3 Structural templates are generated by replacing tokens with linguistic features, and offer text groupings from different aspects. While t_2 uses \$PERSON to find rich celebrities, one can instead get t_4 to explore other queries about “Bill Gates” (<i>e.g.</i> , “how old” in q_3).	5
1.4 Polyjuice generates diverse counterfactuals that experts may miss. For example, to teach the model that negation changes sentiment, experts may augment the training data by rewriting is \rightarrow is not , but they would miss teaching the model is \rightarrow could have been	6
1.5 We design ScatterShot to help humans appropriately specify their desired model behavior in in-context learning. Given an existing state of in-context training examples that is likely <i>under-specifying</i> the intended functionality (A_1), ScatterShot uses the set to drive an LLM (<i>e.g.</i> , GPT-3) to suggest (possibly noisy) annotations to randomly selected unlabeled examples (B_1), filter the ones that are likely to contain not-yet-demonstrated patterns (B_2), such that humans can correct the suggested annotation (B_3), add them back to the in-context training set (A_2), and start over. . .	7
1.6 <i>Chaining</i> makes the human-AI collaboration more transparent and controllable. For example, we decompose a peer review rewriting task into three sub-tasks: identifying each problem, providing suggestions per problem, and composing all the suggestions into one paragraph. Users can inspect and steer AI in each step. . .	8

2.1	The Errudite interface, with (A) model overview; (B) attribute histograms (§2.4, enlarged version in Figure 2.5); (C) filtering panel for users to specify DSL queries (§2.3.1), (D) instance list displaying filtered examples; (E) list of saved groups (§2.3.2 and Figure 2.3) and (F) rewrite rules (§2.3.3). See §2.4 for more details.	12
2.2	An example MC error with the ground truth and the <u>prediction</u> both being “PERSON” entities.	12
2.3	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.)	17
2.4	Updated <u>prediction</u> in response to the rewrite rule <code>rewrite(c, STRING(p(m)) → "#")</code>	18
2.5	The distribution of ENT(g) in group (a) <code>is_entity</code> and (b) <code>is_distracted</code> . The histogram shows the absolute frequency and incorrect/correct ratio for each attribute value.	19
2.6	Rewrite rules inferred from an edit to an individual instance.	20
2.7	Percentage of errors covered by user-defined groups: <i>Boundary</i> ($\mu = 30.9\%$, $\sigma = 10.5\%$) and <i>Multi-sentence</i> ($\mu = 13.5\%$, $\sigma = 8.29\%$). The dispersion of grey ticks shows that users come up with different definitions for groups described by Seo et al. [2017], even when they think they replicated the group faithfully.	22
2.8	Example instances that fall into different user-defined <i>Boundary</i> groups.	23
2.9	Usefulness of function modules in the tool.	24
3.1	Structural templates t generated from a given query q	29
3.2	Tempura interface. (A) The <i>template overview panel</i> lists structural templates automatically selected to represent the dataset’s query distribution. Each row is associated with attributes summarizing their corresponding queries (E, details in Fig. 3.5). (B) The <i>query panel</i> lists individual queries in the dataset. (C) The <i>template traversal panel</i> allows users to navigate related templates. A focused template selected for traversal (F, enlarged in Fig. 3.8) has visual indicators suggesting related templates to explore. Clicking on the traversal options attached to each linguistic feature opens up the neighbor template table (D). The table displays the corresponding parents or children, with the differing linguistic feature highlighted (G). Like in (G), all templates allow previewing the example queries covered by an expanded template. Video demo: https://youtu.be/s_ODGuZU4G8	30
3.3	Queries and their generated templates form a many-to-many relationship. Each query generates multiple templates, and each template covers a different group of queries.	35

3.4	Parent and child templates for “how \$ADJ \$be \$PERSON”, by tweaking its different linguistic features: (A) is a parent template found by merging \$be with other possible LEMMA (e.g., \$do) to \$VERB (POS). Similarly, (B) merges \$PERSON with other possible ENT to get \$NOUN (POS). (C) and (D) are two sets of child templates <i>splitting</i> (C) the \$ADJ and (D) \$PERSON into actual LEMMA forms.	36
3.5	A template’s attributes, summarizing its corresponding query group: (A) the number of queries covered; (B) the error rate; and the label distribution of queries’ (C) ground truths and (D) predictions.	38
3.6	Given a focused template $t = \text{“how } \$ADJ \$be \$PERSON\text{”}$, we can traverse to its parents (up arrows on top of \$be and \$PERSON), or children (down arrow under \$ADJ and \$PERSON). The color density of the arrows indicates how informative a traversal is. Bar charts denote the coverage distribution of the templates affected by the traverse.	39
3.7	The five most representative templates and most frequent query types for the sampled queries in (A) Assistant, and (B) MS MARCO. They have similar query types, but their templates differ after “what \$be \$NOUN”. This indicates that the templates can capture more dataset characteristics than the conventional query types.	44
3.8	Traversing a focused template. The dark arrow in B suggests moving from \$can to its parent form \$VERB.	45
3.9	The ground truth label distributions for queries covered by two closely related neighboring templates, with (B) “how \$do \$-PRON- \$VERB \$NOUN” having a much larger portion of <i>food</i> related queries.	46
3.10	Neighbor template table showing the top four LEMMA (\$make, \$cook, \$play, and \$do) after splitting the second \$VERB in “how \$VERB \$-PRON- \$VERB \$NOUN”.	47
4.1	Overview: (A) given a sentiment analysis instance x , POLYJUICE generates (B) various counterfactuals \hat{x} , which are then (C) selected for downstream use. <i>e.g.</i> , in (D) we select counterfactual explanations that complement a black box explanation: though “great” and “kids” are deemed important, perturbing them may not affect the prediction $f(x) = f(\hat{x}) = positive$, revealing model failures not covered by feature attributions.	57

4.2	(A) POLYJUICE prompt format, which concatenates the original x , the control code, and the \hat{x} (“It is not great for children” converted to an infilling structure). At <i>generation</i> time, POLYJUICE accepts prompts that just include x (Line 1), or optionally with the code and the [BLANK]s (Lines 2–3), and fills in the blanks sequentially with spans separated by [ANSWER]s (Line 4). (B) POLYJUICE allows blanking at different granularities (even the entire sentence), such that Lines 3–4 in (A) can be replaced by Lines 6–7 or 8–9.	59
4.3	(A) An instance in <i>QQP</i> where the model prediction $f(x)$ is <i>Duplicate</i> (=) at 98.2% confidence, with SHAP importance weights for tokens in Q2. Counterfactual explanations complement SHAP with concrete examples and surprising behaviors, e.g., (B) shows that friend \rightarrow woman surprisingly flips the prediction to <i>Non-Duplicate</i> (\neq), despite the low weight on “friend.”	67
4.4	Simulation error rates per condition (higher the better). POLYJUICE- <i>surprise</i> has the highest error rate, indicating these counterfactuals would add the most information to users if displayed.	68
4.5	(A) An <i>NLI</i> case with a <i>Neutral</i> prediction (<u>underlined</u> $f(\hat{x})$ are correct). POLYJUICE generates counterfactual hypotheses conditioned on the negation control code. (B) Generalize perturbations into patterns (Chapter 3). DET \rightarrow no flips 92.8% of predictions from <i>Neutral</i> \rightarrow <i>Contradiction</i>	70
4.6	Perturbing the subject of x in Figure 4.5A through [BLANK], resulting in erroneous predictions for different <i>quantifiers</i> (all should be <u><i>Neutral</i></u>).	71
5.1	An overview of how humans and LLMs can use ScatterShot to iteratively create in-context functions for temporal expression extraction and normalization, with more diverse training sets, and more accurate annotations. The function extracts phrases with temporal meaning from sentences (e.g., “today” in “Today’s service was awesome :3”), and normalizes them into standard formats (“today == 2014-10-18”). Given an existing state of in-context training examples that is likely <i>under-specifying</i> the intended functionality (A_1), ScatterShot uses the set to drive an LLM (e.g., GPT-3) to suggest (possibly noisy) annotations to randomly selected unlabeled examples (B_1), filter the ones that are likely to contain not-yet-demonstrated patterns (B_2), such that humans can correct the suggested annotation (B_3), add these correct input-output pairs back to the in-context training set (A_2), and start over.	76

5.2	The ScatterShot interface, with (A) task description, (B) existing in-context examples, and (C) candidate examples awaiting human inspection. (D) Users can make edits to LLM outputs, sort the candidates into positive demonstrative examples (+), negative ones (-), or just not include the candidate (O). The description and the examples get transformed to raw text prompts. One set of in-context example produces multiple prompts depending on how the examples are ordered; (E) shows a prompt with one possible ordering.	80
5.3	A high level overview of ScatterShot’s slice-based sampling. We use the data status in the $i - 1$ -th iteration to perform sampling for the i -th iteration: (A) we use the already LLM- or human- assessed examples to infer data slices represented as templates and estimate how the current in-context function performs on different slices;(B) we use these templates to partition the unlabeled data space into various groups (phrases that match the templates are <u>underlined</u>); (C) we sample unlabeled examples based on their corresponding slice size and estimated accuracy, to prioritize sampling from more uncertain and larger slices; (D) we run the current in-context function on the selected inputs, and deem the function correct, if prompts with different in-context example orders produce the same output. An example will only be surfaced to the interface for human inspection if the function is incorrect on it (annotated with the orange dot).	81
6.1	A walkthrough example illustrating the differences between <i>no-Chaining</i> (A) and <i>Chaining</i> (B), using the example task of writing a peer review to be more constructive. With a single call to the model in (A), even though the prompt (italicized) clearly describes the task, the ● <i>generated paragraph</i> remains mostly impersonal and does not provide concrete suggestions for all 3 of Alex’s presentation problems. In (B), we instead use an LLM Chain with three steps, each for a distinct sub-task: (b1) A Split points step that extracts each individual presentation ● <i>problem</i> from the ● <i>original feedback</i> , (b2) An Ideation step that brainstorms ● <i>suggestions</i> per problem, and (b3) A Compose points step that synthesizes all the problems and suggestions into a final ● <i>friendly paragraph</i> . The result is noticeably improved.	93
6.2	An example of how to create an LLM step using a prompt template (A), using the <i>Ideation</i> step of the peer review writing scenario (from Figure 6.1) as an example. For the peer review scenario, the <i>Ideation</i> operation takes in a problem (<i>e.g.</i> , too much text) as input, and produces suggestions for improvement as output, but the prompt template allows the <i>Ideation</i> operation to take in any custom inputs and outputs. The template includes placeholders for the input (prefix-1), output (prefix-2), and (optional) few-shot examples. (B) shows the actual prompt after filling in the placeholders in the prompt template.	104

6.3	An overview of the interface, reflecting the peer review rewriting example in Figure 6.1. It consists of (A) a <i>Chain view</i> that depicts the high level Chaining structure, and (B/C) a <i>Step view</i> that allows for refining and executing each LLM step. The interface facilitates tracking the progress of the LLM Chain. For example, when moving from step 2: <i>Ideation</i> (B) to step 3: <i>Compose Points</i> (C), the previously generated presentation problems and suggestions become inputs for the final paragraph. A demonstration is available at https://youtu.be/QFS-1EW1vMM	106
6.4	The LLM Chain for flashcard creation, with: (A) An Ideation step that brainstorms the ● <i>types of interactions</i> that we might encounter when ● <i>visiting a given city</i> (Paris), (B) Another Ideation step that creates a list of ● <i>English examples</i> for each ● <i>interaction type</i> , and (C) A Rewriting step that translates each ● <i>English example</i> into ● <i>French</i>	111
6.5	Participants’ ratings in the form of seven-point Likert scale questions (details in Appendix C.2.1), with 95% confidence intervals. Using Chaining, participants felt they produced results that better <i>matched the task goals</i> , and that the system helped them <i>think through</i> the task. They also found Chaining more <i>transparent</i> , <i>controllable</i> , and <i>collaborative</i>	114
6.6	Distribution (based on the logged interactions) of how participants interacted with the prompts and model outputs, with and without chaining. (A) They made more <i>edits</i> in Chaining (compared to just repeatedly running the model), and (B) They tended to <i>curate</i> model outputs, rather than either deleting (<i>undoing</i>) them entirely or manually <i>creating</i> new content.	114
6.7	An example for Chaining-based VegaLite bug fixing (simplified; the full Chain is in Appendix C.3). (A) We first rewrite the ● <i>JSON format specs</i> into ● <i>natural language descriptions</i> to make it more parsable, then (B) classify the descriptions to ● <i>validate design constraints and suggest fix</i> and (C) finally rewrite the ● <i>final spec</i> based on the suggested fix. While the LLM generates the fix in d_1 , users may also choose to produce d_2 and d_3 , both of which can fix the ● <i>validated issue</i> just as effectively.	123
6.8	An example of Chaining-based assisted text entry (the full Chain is in Appendix C.3). To produce better full sentences, we classify the input text to switch between expanding shorthands (through Rewrite) and auto-completing phrases (through Generation). By wrapping the complex Chaining logic in a simple text field, we provide intuitive interactions for end users.	125
A.1	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	177

A.2	VQACounting improves much more on <code>how_many_NOUN</code> , compared to <code>how_many_ADJ</code> , though many fewer instances follow the latter pattern.	177
A.3	Extracting the POS tag for the token immediately after “how many”, we notice most instances follow a “ <code>how many NOUN</code> ” pattern.	178
A.4	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.	178
A.5	An instance with 8 distinct ground truth annotations, and negligible inter-annotator agreement.	178
A.6	Two instances suspected to be wrong due to tokenization.	179
A.7	Three types of changed predictions on instances generated with <code>add_space</code>	179
A.8	The prediction length <code>length(p(m))</code> is much longer for (a) only “why” questions than for (b) only “what” questions.	180
A.9	A “why” question where the model ignored the apparent hint “because.”	180
A.10	The illustrating example we used in the chapter; we repeat it here to explain our programming-by-demonstration heuristics. The scenario here assumes “John” is selected by a user.	181
A.11	Rewrite rules inferred from an edit on an individual instance.	183
B.1	A sample labeling task: The crowdworkers annotate three counterfactuals based on their validity and class label, with respect to the original instance.	192
B.2	The accuracy trend on two <i>Sentiment</i> datasets, as the total training datasize ($m + n$) varies. The blue line shows an augmentation of $m = 2k$ counterfactuals, and the blue one represents the corresponding m -baseline. Though the counterfactuals remains useful on datasets like SemEval across all $m + n$, it appears too many counterfactuals may be harmful (Amzbook).	193
B.3	A sample explanation task for §4.4	194
B.4	The <i>NLI</i> model cannot perform the actual counting when the exact number is missing from P	196
C.1	The chain for acronym expansion. The steps include: (A) An ideator that brainstorms various ● <u>unique traits</u> for ● <u>the concept</u> (crowdsourcing). (B) For each trait, a generator creates a related ● <u>metaphor</u>	201

- C.2 The LLM Chain for visualization bug fixing (in VegaLite). The stages include: (A) A **Rewrite** step that transforms the ● *json format VegaLite spec* into ● *natural language description*, so to eliminate noise from data. (B) An **Information Extraction** step that locate ● *related visualization rules*. (C) A **Classification** step that verifies the description as either ● *valid or invalid* (with concrete errors and fixes). (D) A **Rewriting** step that generates ● *the fixed VegaLite spec* based on the ● *validity reasons*. 205
- C.3 The LLM Chain for assisted text entry. The stages include: (A) A **Classification** steps that detects whether a ● *given sentence* ● *contains a shorthand or not*. (B) If there exists certain shorthand, a **Rewriting** step expands it, so we arrive at the ● *expanded sentence* which can become the context for additional shorthand inputs. For “LTSG”, it can be “Let’s go” or “Let’s get”, which relies on human selection. (C) Otherwise, a **Generation** step auto completes ● *the sentence*. 206

ACKNOWLEDGMENTS

First, I would like to start by thanking my advisors, Jeffrey Heer and Dan Weld, for being impeccable mentors on every measure. I came to them when I was a clueless newbie without a research agenda, and they generously and patiently gave me the time and freedom to explore my interests. As if by magic, they were always able to guide me and teach me how to form bigger pictures, nail technical details, as well as share results, without stressing me out. During my PhD journey, I have never felt abandoned, constrained, or pressured, which I later realized is a rare fortune I should not have taken for granted. In addition to the research training, I also wanted to thank them for being wonderful human beings. The mental support they provided me (e.g., by reassuring me in every annual review meeting), and the trust they showed in me, is a great gift. I am not exaggerating when I say that without their constant encouragement, I would never have considered a faculty position. Of course, their support takes many physical forms as well. They went out of their way to make sure I can “stand on the shoulders of giants,” nominating me for fellowships and internships before I even knew to look, helping me make connections, and recommending me for all kinds of workshops, panels, and career opportunities. I believe that I am shaped by the environment I’m in, and I owe half of my accomplishments to the best advisors I could ever hope to find. My students are lucky if I can be half as good as my advisors!

Second, I would like to thank Marco Tulio Ribeiro. After closely collaborating with him for three years (and on four papers), I practically consider him to be my comrade-in-arms *and* third advisor. Junior advisor Marco undoubtedly has had a big influence on my research career. He provided me with countless, practical tips on how to design a project, manage meetings, and manage my time. Additionally, he never ignored my rambling, always treated my issue as if it were his own, and offered valuable suggestions throughout the process. He was also one of the first to recognize

me as an NLP person with an HCI mindset (rather than the other way around), which helped me build my confidence towards some serious ACL publications. I still remember him telling me when we met that “we should terminate the collaboration anytime if it didn’t work out”, and, well I’m glad it worked out!

Along the line of advising, I am also thankful to the rest of my committee members, Noah Smith and Mari Ostendorf, for their feedback and questions that helped improve this thesis. Going a bit back in time, people from HKUST VisLab, including Huamin Qu, Weiwei Cui, Conglei Shi, and Yingcai Wu were the first batch who provided an opportunity for me to try out research. Thank you, for opening up a magical world for a young undergraduate student.

Besides (semi-)advisors, collaborations were a major factor in my PhD. I was lucky enough to have done four internships throughout my journey, with Saleema Amershi, Ece Kamar, Bongshin Lee, Marco Tulio Ribeiro at Microsoft Research, Chris DuBois, Kayur Patel, Kanit (Ham) Wongsuphasawat, Donghao Ren and Charlie Maalouf at Apple, and Carrie Cai and Michael Terry at Google PAIR. Partly because I struggled to effectively complete my internship projects, but partly because my mentors and collaborators were too generous and were willing to keep me on the team, all of my supposedly three month internship eventually led to collaborations for at least eight months. I gained unique knowledge from each of my mentors, for example Ham’s focus on algorithmic designs, Carrie and Mike’s sharp eyes on impactful research, and Ece and Saleema’s ability to gather resources needed for a team. I also appreciate the opportunity to get a glimpse into all these incredible tech companies and learn how to conduct research with users in mind, how to collaborate with engineers and designers, and how to translate research prototypes into actual, usable tools. These experiences changed my thinking of research and impact.

Of course, I learned just as much from all my other close collaborators, some of whom are now my closest friends: Alexis Ross, Hao Peng, Matthew Peters, Matt Gardner, Gagan Bansal, Joyce Zhou, Raymond Fok, Besmira Nushi, Carlos Guestrin, Sameer Singh, Ellen Jiang, Hua Shen, Kenneth Huang, Jiao Sun, Yue Jiang, Xi Victoria Lin, Diyi Yang, Dakuo Wang, Toby Jia-Jun Li,

Alison Smith-Renner. I enjoy collaboration the most because it gives me a chance to learn about working styles and research topics that I wouldn't have thought of otherwise. Thanks to you all for providing me with this opportunity.

Even more broadly, I am grateful to the entire research community of Interactive Data Lab, Human-AI Lab, UW Allen School, HCI, and NLP. It has always been a pleasure to work with smart, friendly, and collaborative people. I survived my PhD because amazing mentors and peers always selflessly gave me much needed guidance, but also because people were compassionate enough to normalize imposter syndrome. Knowing that everyone feels insecure from time to time is immensely comforting. Being on the edge of two fields, I'm grateful that both communities are open-minded and I've never felt out of place, regardless of whether I'm at CHI or ACL. My Seattle support group is equally essential; I will forever remember the food, the grocery trip, the leisurely chitchats with various friends like Yuehui Wang, Jim Chen, and Sitong Zhou, just to name a few (I would run out of room if I listed all the names).

Last but not least, I want to express my gratitude to my family, especially my parents Daihong Wang and Jianjun Wu, for their unconditional love and support. Despite not being into computer science, they remembered all my deadlines, participated in all my life decisions (from where to eat tonight to where to work in the summer), and attended my online talks regardless of timezone differences. I cannot imagine a better family than ours. I selfishly chose a path that took me far from home, and I always feel a pang of guilt when I think of all the things I temporarily left behind. For what it's worth, I hope they will be proud of who I become.

DEDICATION

To my dear parents, Daihong Wang and Jianjun Wu,
I don't know what I would do without you.

Chapter 1

INTRODUCTION

Research in Artificial Intelligence (AI) has advanced at an incredible pace, to the point where it is making its way into our everyday lives, explicitly and behind the scenes. In spite of their rapid progress, many AI models hide nuanced weaknesses: AI models, regardless of how "accurate" they are, engage in inconsistent behavior on similar inputs [Ribeiro et al., 2019, Gardner et al., 2020], and can struggle with certain types of input distribution [Wexler, 2017]. For example, a simple synonym switch can easily break sentiment analysis models and lead to wasted client investments [Ribeiro et al., 2020], and Twitter's image cropping algorithm favors white people as image previews rather than black people [Shen et al., 2021], which amplifies social biases. As a result, deficient models cannot be trusted for use in the real world.

Obviously, to protect the end-user experience, we should only deploy models that are trustworthy, while iterating on those that have obvious weaknesses. However, due to the complexity of state-of-the-art models, our current model development process cannot capture unwanted model behaviors. For example, we represent model quality with holdout accuracy [Wang et al., 2018a], but the metric and dataset are flawed: accuracy scores are often too sparse to reflect our expectations of fairness or robustness [Wang et al., 2021c], and training/validation data have too many annotation artifacts and biases [Gururangan et al., 2018b] to be representative of real-world scenarios. Consequently, the scores usually overestimate models' true performance [Recht et al., 2019]. Developers have tried to inspect their models more closely (e.g., by creating in-house benchmark datasets [Bansal et al., 2019a]), but have yet to succeed. Even commercialized models (e.g., the sentiment classifier model and image cropper mentioned above) that have allegedly undergone extensive bug discovery and fixing can fall victim to undesired and unknown bugs.

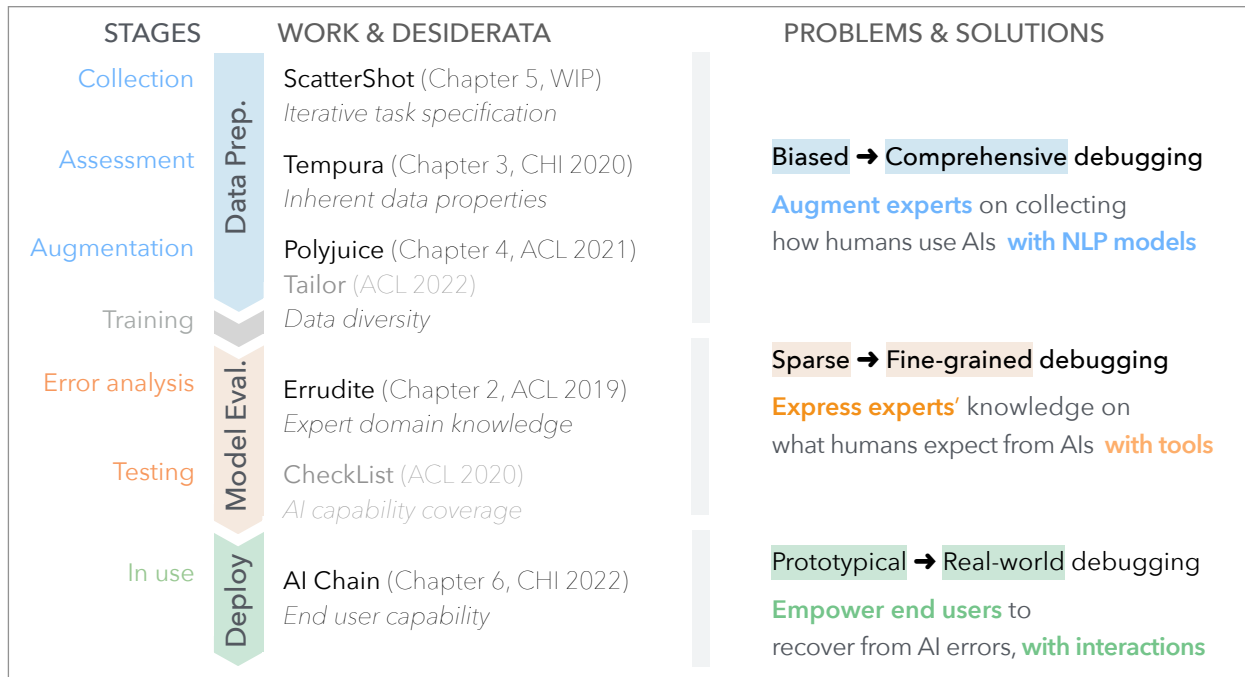


Figure 1.1: Thesis overview: We present how the human perspective is missing in three major model development stages, and address the issues by building build tools to interactively help humans debug and correct models.

This poses an essential question: how do we create opportunities for humans to debug and correct the models, so that these models are more aligned with real-world use cases? In this dissertation, we examine pitfalls in model development that hinder humans from identifying, improving, and coping with imperfect models, and present new frameworks, NLP models, and interactive interfaces to address those problems.

1.1 Thesis Overview

Why does the gap between “accurate models” (those that make it to the top of leaderboards [Rajpurkar et al., 2016b]) and “successful models” (those that support real world use cases) exist? Here, we describe three problems that hinder the status-quo model debugging effort, as well as the solutions we propose. First, relying solely on *standard evaluation paradigms* (e.g., computing model accuracy) can lead to **sparse debugging**, *i.e.*, experts are unable to fully comprehend models’

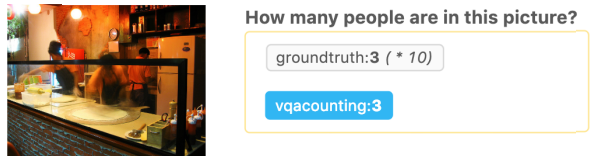
nuanced strengths and weaknesses. In response, we need to give experts the flexibility to inspect models using their domain knowledge (Chapter 1.1.1). Second, allowing experts to solely rely on their *domain knowledge* can lead to **biased debugging**, *i.e.*, experts may mistakenly over-emphasize (or overlook) certain model use cases. As such, we need to assist experts in overcoming their potential omissions on the use case distribution (Chapter 1.1.2). Third, relying solely on *experts* can lead to **“prototypical” debugging**, *i.e.*, even carefully developed models are guaranteed to make mistakes *by design* [Hong et al., 2021], and can be frustrating to use if the error can not be corrected. Therefore, we should also involve end users, and help these people to interpret and control AI models as they collaborate with them in real time (Chapter 1.1.3). As shown in Figure 1.1, our efforts are spread across the entire model development cycle, from data preparation to model evaluation to post-deployment.

1.1.1 Express experts’ knowledge in model analysis, for *finer-grained* debugging

While we track model improvements with standard metrics such as F1 or accuracy [Wang et al., 2018a], these evaluation metrics are often too sparse to capture humans’ expectations, *e.g.*, models should be right for the right reasons and should not rely on spurious features [Kaushik et al., 2020, Gardner et al., 2020], nor should they strive for high performance on the held-out set at the expense of smaller data slices [Zhang et al., 2018a]. To determine when, how, and why models fail, we must perform additional model analysis and testing. However, because NLP models are typically complex, and unstructured text data can be difficult to filter semantically, experts rarely review more than a few examples at a time. Unfortunately, these ad hoc observations can result in confirmation bias and spurious conclusions, and hinder experts from improving model quality [Wu et al., 2019b].


How can we ensure that we evaluate and analyze models in a way that reflects our expectations of them? We designed interactive tools and novel NLP models that **enable experts to express their expectations on models**. In particular, we built Errudite [Wu et al., 2019a] (Chapter 2) to facilitate model error analysis. I identified and implemented two essential building blocks, as shown in Figure 1.2: First, Errudite allows systematic grouping of relevant instances and thus scales observations beyond random spot checks; Second, Errudite supports counterfactual rewriting,

A Hypothesis: Model cannot count objects with ADJs



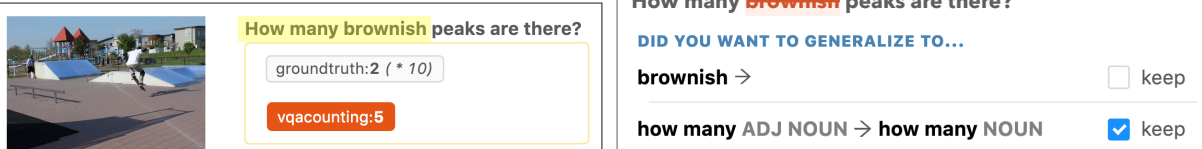
How many people are in this picture?
groundtruth:3 (* 10)
vqaounting:3

B Systematic *grouping* with filtering rules



has_pattern(question, pattern="ADJ")
starts_with(question, pattern="how many ADJ")
"How many peaks are in brown?"

C Systematic *counterfactual analysis* with rewrite rules



How many **brownish** peaks are there?
groundtruth:2 (* 10)
vqaounting:5

How many **brownish** peaks are there?
DID YOU WANT TO GENERALIZE TO...
brownish → keep
how many ADJ NOUN → how many NOUN keep

Figure 1.2: Errudite example. This tool enables scalable and testable error analysis through *systematic grouping* and *counterfactual rewriting*. (A) A visual question answering model predicts “How many people...” correctly but “How many brownish...” incorrectly. Experts suspect the adjectives make a difference. (B) They **scale up the observation**, by building groups of questions that contain \$ADJectives, or start with \$“How many ADJ”. (C) They also **test the root error cause** with rewrite rules that answer: “*If the adjectives were not there, would the model predict correctly?*”

which surfaces root error causes by testing what-if scenarios. We validate our approach with a user study, and confirm that by transforming expert domain knowledge into actionable analysis scripts, Errudite enables users to test and revise their prior beliefs on models in a reproducible way with less effort. As a result, developers and researchers have adopted it for their own NLP models, and shared reproducible insights with the broader NLP community (*e.g.*, researchers working on relation extraction and an internal team at Apple doing question answering).

1.1.2 Augment experts in data collection, for **comprehensive debugging**

Errudite, while effective, focuses on error analysis and only supports evaluating models once they have been built. However, even in the early stages of model development, we encounter data collection problems. Due to almost inevitable artifacts of the annotation process, task framing, or design decisions [Gururangan et al., 2018b, Liu et al., 2021a, Geva et al., 2019, Schwartz et al., 2017], most (if not all) existing datasets have subjective and noisy labels [Chen et al., 2021a], or contain systematic gaps (*e.g.*, differences from the real-world distribution that can be exploited by models to achieve artificially high test accuracy.) [Gardner et al., 2020]. As a result, models

trained on these datasets learn to exploit these biases in order to boost performance on the test set sampled from the same distribution, which means they model the datasets but not the tasks [Wu et al., 2022c]. Thus, analyzing and fixing the training and evaluation data is an essential part of the model development process [Hohman et al., 2020, Sambasivan et al., 2021].

The aforementioned grouping and counterfactual rewriting are also fundamental in the data fixing region; However, relying solely on experts’ prior beliefs or domain knowledge (as in error analysis) can introduce unnecessary bias. For example, when experts sanity check training data quality, they need groupings to detect conflicting labels between similar instances. Here, handcrafted filtering rules may be subjective and produce unrepresentative groups. To mitigate human bias, we design various tools that extend the two building blocks with automated approaches (NLP models, automatic mining algorithms), such that they can **assist experts in collecting data on how humans use AIs**.

First, for the aforementioned data assessment scenario, we developed Tempura (Chapter 3), which automatically exposes implicit properties in both academic and industrial datasets. With a new grouping method called *structural templates*, Tempura mines and ranks text groups from a dataset, which helps examine data from different structural aspects and at different granularities (Fig. 1.3). The grouping is achieved *without* relying on manual queries, and therefore mitigates potential developer biases. Tempura further facilitates the analyses by showing overviews first (i.e., a subset of representative groups) and providing details on demand (i.e., recommending related groups likely to yield further explorations.) User studies also showed that Tempura supports both data exploration and error analysis, helping practitioners examine the distribution of the data, find labeling errors, and discover model error patterns and outliers in a query dataset.

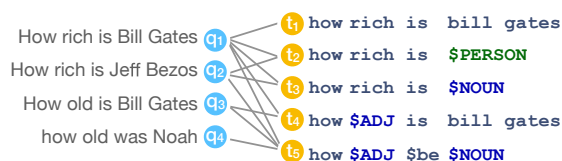


Figure 1.3: Structural templates are generated by replacing tokens with linguistic features, and offer text groupings from different aspects. While t_2 uses $\$PERSON$ to find rich celebrities, one can instead get t_4 to explore other queries about “Bill Gates” (e.g., “how old” in q_3).

Second, to disentangle spurious and robust features via counterfactual data augmentation (*e.g.*, to solely highlight the impact of *negation* independent of other features using the original and perturbed sentences in Figure 1.4), we proposed Polyjuice (Chapter 4), a language-model-based generator that automatically produces diverse counterfactuals. In a variety of domains, not only did the Polyjuice generator compensate for human omissions (Figure 1.4), but its generated counterfactuals successfully improved model generalization. In this way, we were able to fix models by removing spurious correlations in data.

Finally, in Chapter 5, we extend our observation in Polyjuice that generators can be inspirational, and build ScatterShot to help humans *iteratively refine their task definition during data collection*. With the in-context learning capabilities of Large Language Models (LLM) like GPT-3 [Brown et al., 2020], developers can tailor an LLM for their specific tasks with only a dozen demonstrative examples [Brown et al., 2020]. Yet, this lightweight adaptability also makes example selection critical – if developers omit corner cases, the task quality can easily be affected [Liu et al., 2022]. To help humans craft diverse and clean in-context learning examples, we design a human-LLM collaboration mechanism shown in Figure 1.5, where LLMs help function creators identify diverse examples and avoid repetitive patterns. In preliminary experiments we find that participants were able to provide more feedback and get inspired by LLM outputs, eventually resulting in better coverage of corner cases in their in-context learning tasks.

1.1.3 Empower interactions with deployed AI models, for *real-world* debugging

-> The previous two parts mostly focus on aiding quality improvement of NLP systems before deployment. Because these systems rely on probabilistic models, errors are still unavoidable even

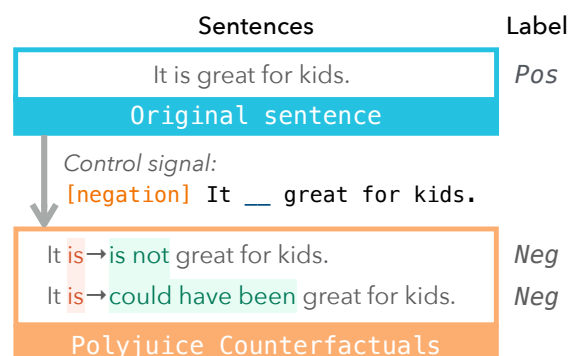


Figure 1.4: Polyjuice generates diverse counterfactuals that experts may miss. For example, to teach the model that *negation* changes sentiment, experts may augment the training data by rewriting *is* → *is not*, but they would miss teaching the model *is* → *could have been*.

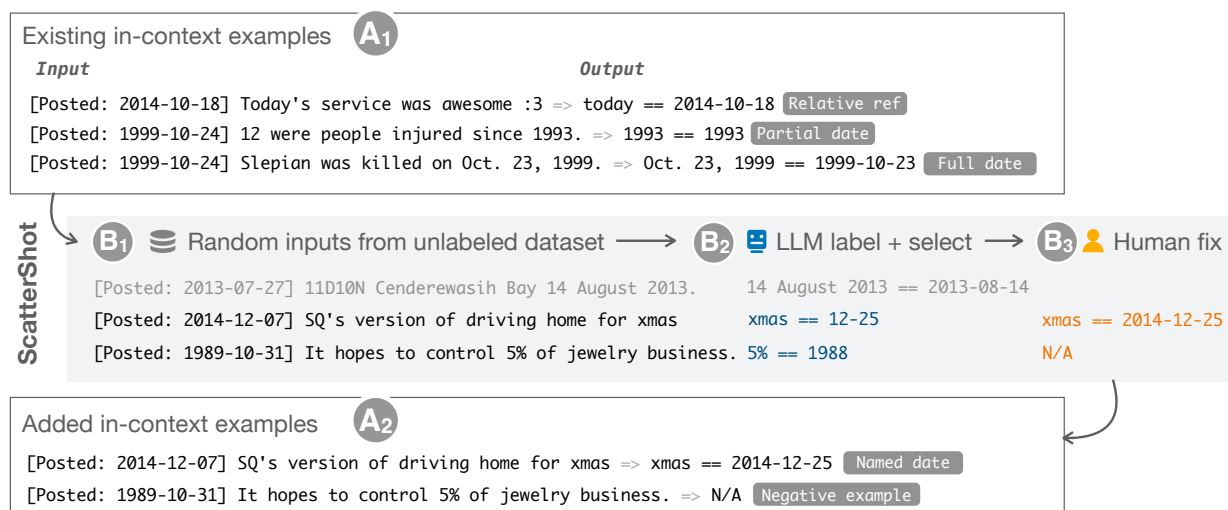


Figure 1.5: We design ScatterShot to help humans appropriately specify their desired model behavior in in-context learning. Given an existing state of in-context training examples that is likely *underspecifying* the intended functionality (A₁), ScatterShot uses the set to drive an LLM (e.g., GPT-3) to suggest (possibly noisy) annotations to randomly selected unlabeled examples (B₁), filter the ones that are likely to contain not-yet-demonstrated patterns (B₂), such that humans can correct the suggested annotation (B₃), add them back to the in-context training set (A₂), and start over.

after careful model designs [Hong et al., 2021]. In other words, end users still inevitably face, and have to collaborate with, imperfect AIs. Such scenarios beg the question: Can we provide users with assistance in interpreting and improving model output when they do not understand the underlying structure and cannot alter the model? **We design interaction technologies to help users recover from errors they see as they collaborate with AIs.**

For example, we created AI Chaining (Chapter 6) to improve interaction between end users and LLMs. Unlike ScatterShot where LLMs lead to improve in-context learning generalizability by suggesting and labeling corner cases, AI Chain focuses on letting *end users* lead LLMs towards their own desired outcomes through interactions. We observed in pilot studies that end users struggled to debug and improve their arbitrary instruction prompts, and in response, we proposed *Chaining* multiple LLM runs together, *i.e.*, decomposing an overarching task into a series of highly targeted sub-tasks, mapping each to a distinct LLM step, and using the output from one step as an input to the next. These steps naturally expose intermediate checkpoints and control knobs to end users,

helping them pinpoint seemingly global errors to a local cause. For example, thanks to the *Ideation* step in Figure 1.6, Chaining lets users customize which suggestions to include in the final paragraph — a function that does not exist otherwise. As a result, Chaining significantly enhanced system transparency and user control, and a sense of collaboration. We later deployed Chaining as an internal tool at Google to support rapid prototyping of LLM-infused applications.

1.2 Thesis Statement

We demonstrate how the *human perspective* is consistently absent across all stages of machine learning model development, and how we strive to include people in the equation using a *data-centered approach*. For example, by systematically grouping and rewriting data, we help experts escape from ad hoc observations, and rigorously compare models’ actual behaviors against their expected behaviors. By augmenting existing data, we enable experts to enhance model training and testing signals beyond their own prior knowledge, mitigating human bias in model debugging and correction. Additionally, by decomposing complex tasks into more transparent and controllable subtasks, we help end users interact better with deployed models. We believe that since data is a common communication medium for humans and undergirds all AI models, it is an ideal bridge between humans and AI. Our thesis is that:

When given strategies and tools to interactively massage (partition, perturb, and decompose) data throughout the machine learning model development stage, developers and end

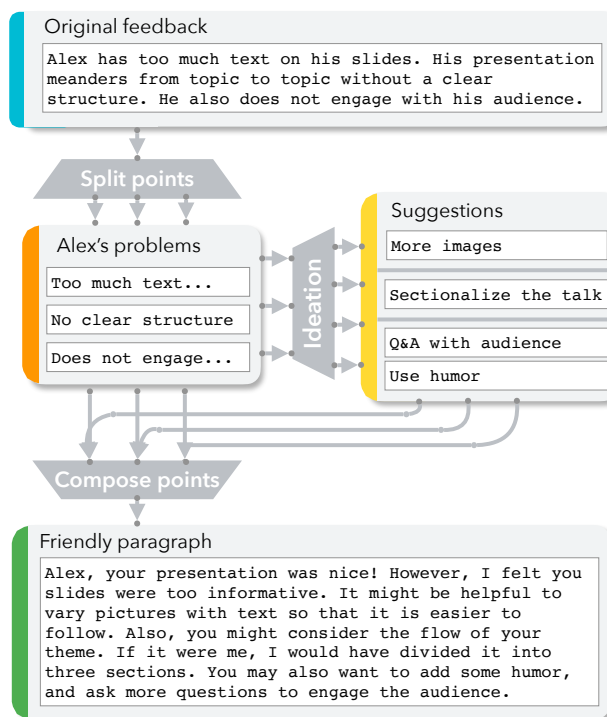


Figure 1.6: *Chaining* makes the human-AI collaboration more transparent and controllable. For example, we decompose a peer review rewriting task into three sub-tasks: identifying each problem, providing suggestions per problem, and composing all the suggestions into one paragraph. Users can inspect and steer AI in each step.

users can debug and correct AI models in a more comprehensive, less biased, transparent, and controllable way.

1.3 Prior Publications and Authorship

Although I am the primary author of the research detailed in this dissertation, it is also the product of years of collaboration with my primary co-advisors, Jeffrey Heer and Daniel S. Weld, and my collaborators at Microsoft Research, Google Research, and Apple. Errudite and Polyjuice (Chapters 2, 4) are in collaboration with Marco Tulio Ribeiro, Dan Weld, and Jeffrey Heer, and are based on publications appeared in ACL 2019 [Wu et al., 2019a] and ACL 2021 [Wu et al., 2021] respectively. Scattershot (Chapter 5) has a similar collaboration structure and at the time of thesis writing is work-in-progress. Tempura (Chapter 3) appeared at ACM CHI 2020 [Wu et al., 2020], and was based on my internship at Apple, with Kanit Wongsuphasawat, Donghao Ren, Kayur Patel and Chris DuBois. Finally, AI Chains (Chapter 6) appeared at CHI 2022 [Wu et al., 2022b], and was based on my internship at Google Research, with Carrie J. Cai and Michael Terry. To reflect my collaborators' contributions, I use the first-person plural in these chapters.

Chapter 2

HELPING EXPERTS EXPRESS DOMAIN EXPERTISE FOR SCALABLE AND REPRODUCIBLE ERROR ANALYSIS

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 chapter 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.

This work was done in collaboration with Marco Tulio Ribeiro, Dan Weld, Jeffrey Heer, and was originally published at ACL 2019 [Wu et al., 2019a].

2.1 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., 2018, 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 (§2.5) 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. Figure 2.2 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

¹The full list of papers is provided in Appendix A.3.

via counterfactual analysis (e.g., by removing the *distractor*).

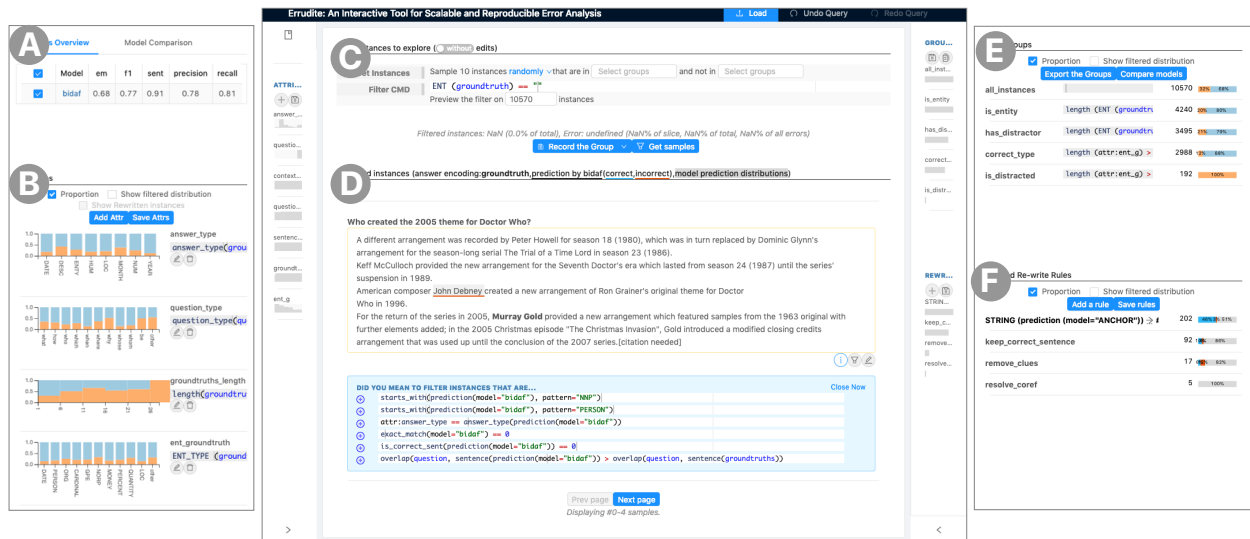


Figure 2.1: The Errudite interface, with (A) model overview; (B) attribute histograms (§2.4, enlarged version in Figure 2.5); (C) filtering panel for users to specify DSL queries (§2.3.1), (D) instance list displaying filtered examples; (E) list of saved groups (§2.3.2 and Figure 2.3) and (F) rewrite rules (§2.3.3). See §2.4 for more details.²

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

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.

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

Figure 2.2: An example MC error with the **ground truth** and the prediction both being “PERSON” entities.

²Video demo: <https://youtu.be/Di15i0AYyu8>.

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 2.2), 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 (§2.3). At each step in the sequence, Errudite helps users inspect and refine their hypotheses in real time with interactive visualizations (Figure 2.1) and query suggestions based on programming-by-demonstration (§2.4). We validate our methodology and Errudite via a user study (§2.5), 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 chapter for easy replication.

2.2 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). 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., 2017] on SQuAD v1.1 [Rajpurkar et al., 2016b] in the rest of the chapter. 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. BiDAF³ is a hierarchical multi-stage end-to-end neural network. It has been widely referenced as a strong baseline model [Wang et al., 2018b, Clark and Gardner, 2018]. Because both SQuAD and BiDAF are common in MC, experts can test and verify prior beliefs about model strengths and weaknesses.

2.3 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.

2.3.1 Precise and Reproducible Hypotheses

Function Name	Definition	DSL Code and resulting Output Values
<code>sentence,</code> <code>token</code>	Extractors for desired spans from targets — sentences or sub-phrases.	<code>sentence(g)</code> → For the return of... <code>token(c, pattern="PERSON")</code> → [John, ...]
<code>exact_match,</code> <code>f1,</code> <code>is_correct_sent</code>	Performance functions that measure different levels of correctness.	<code>f1(m) == 0</code> , <code>exact_match(m) == 0</code> <code>is_correct_sent(m) == False</code>
<code>length</code>	Length of the target.	<code>length(q) == 9</code> , <code>length(g) == 2</code>
<code>POS, ENT, LEMMA</code>	Tokens in the target that have certain patterns of POS tags, named entity, etc.	<code>ENT(g, get_root=True) == "PERSON"</code>
<code>has_pattern,</code> <code>starts_with,</code> <code>ends_with</code>	To check whether the target contains certain pattern. <code>pattern</code> automatically detects queries on POS tags and entity types.	<code>starts_with(q, pattern="who VBZ") == True</code> <code>has_pattern(g, pattern="PERSON") == True</code>
<code>overlap(t1, t2)</code>	The ratio of <code>t1</code> tokens that also occur in <code>t2</code> .	<code>overlap(q, sentence(g)) == 0.25</code>

Table 2.1: Definitions for a subset of attribute extractors, including sample values from the example in Figure 2.2.

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*

³We used the implementation from AllenNLP [Gardner et al., 2018]: <https://allennlp.org/models>.

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 2.1 provides an abridged listing of extractors, with example values from Figure 2.2.⁴

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

⁴A complete list is available in Appendix A.4.

between research groups, and automatically applied to new datasets and models.

2.3.2 Analyze All Relevant Instances

Random spot checking of errors can lead to confirmation bias and spurious conclusions [Rondeau 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 2.2. This is a hypothesis independently raised by four out of ten user study participants (§2.5).⁵

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

```

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

```

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 `c`ontext than in the `g`round 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 2.3.

⁵Participants tested the hypothesis for a specific entity type (numbers). We present a more general case here.

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.





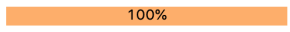
	(a)	(b)	(c)
<code>all_instances</code>	N/A	10570	
<code>is_entity</code>	Line 1	4240	
<code>has_distractor</code>	Lines 1-3	3495	
<code>correct_type</code>	Lines 1-4	2988	
<code>is_distracted</code>	Lines 1-5	192	

Figure 2.3: Saved groups with their (a) manually created and semantically meaningful names, (b) query definitions, and (c) sizes and error rates (orange indicates errors, blue indicates correct predictions.)

2.3.3 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))→"#")`.

The results from the rewrite rule are presented in Figure 2.4. 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.

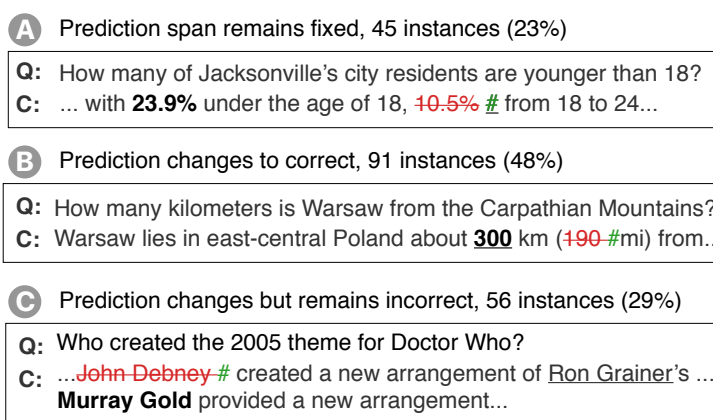


Figure 2.4: Updated prediction in response to the rewrite rule `rewrite(c, STRING(p(m))→"#")`.

2.4 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 pre-

viously 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 2.5 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 2.5(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 2.5(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 2.5(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.

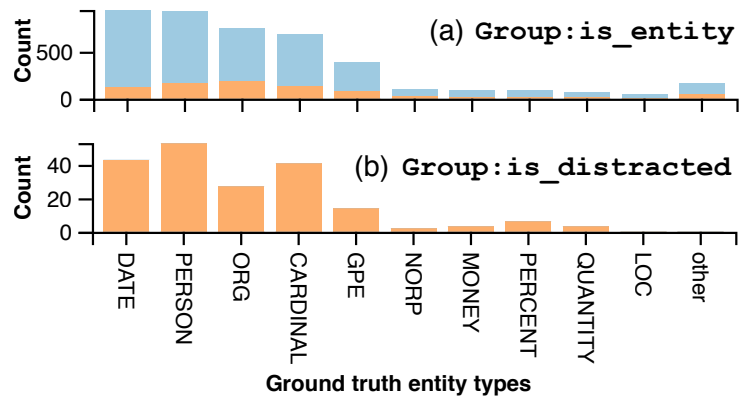


Figure 2.5: The distribution of $ENT(g)$ in group (a) `is_entity` and (b) `is_distracted`. The histogram shows the absolute frequency and incorrect/correct ratio for each attribute value.

For example, selecting “John” in Figure 2.1 (or Figure 2.2) triggers the following suggestions:

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

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 2.6 shows an example in which various suggestions are displayed after a user rewrites an instance by changing “Who” to “What person.” Appendix A.2 provides a

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

DID YOU WANT TO GENERALIZE TO...	
who → what person	<input type="checkbox"/> keep
who created → what person created	<input type="checkbox"/> keep
who VERB → what person VERB	<input checked="" type="checkbox"/> keep
who created the → what person created the	<input type="checkbox"/> keep
who VERB DET → what person VERB DET	<input type="checkbox"/> keep

Figure 2.6: Rewrite rules inferred from an edit to an individual instance.

more detailed description of our searching and ranking criteria.

Layout. The UI (Figure 2.1) 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.

2.5 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.1.3), participants were asked to perform two tasks: **Replication** (§2.5.1), in which they attempted to reproduce the error analysis from Seo et al. [2017]; and **Exploration** (§2.5.2), 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.

2.5.1 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 (2) to assess the reproducibility of current ad-hoc error analysis methods. Seo et al. [2017] 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. [2017]. 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. [2017]: 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 ($\mu = 4.3$, $\sigma = 0.64$), but ambivalent towards *Paraphrase* ($\mu = 3.1$, $\sigma = 0.54$). Participants’ comments indicated the ambivalence did not come from Errudite: 6 participants disagreed with the example given by Seo et al. [2017], 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 ($\mu = 2.8$, $\sigma = 1.32$), 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* ($\mu = 4.8$, $\sigma = 0.60$), yet the groups they produced were wildly different (Figure 2.7). While users were able to express what they thought was meant by “imprecise

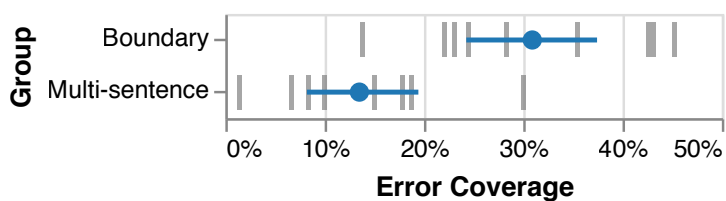


Figure 2.7: Percentage of errors covered by user-defined groups: *Boundary* ($\mu = 30.9\%$, $\sigma = 10.5\%$) and *Multi-sentence* ($\mu = 13.5\%$, $\sigma = 8.29\%$). The dispersion of grey vertical ticks shows that users come up with different definitions for groups described by Seo et al. [2017], even when they think they replicated the group faithfully.

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 2.8 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 2.8(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 2.8(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. [2017]—even when they thought they did—due to the ambiguity inherent in manual grouping.

2.5.2 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 $\mu = 2.1$ findings ($\sigma = 0.94$). Some insights *confirmed prior hypotheses* about BiDAF more formally, increasing users’ confidence. For example, one user created a group to verify that mistakes frequently occur when there is significant overlap between

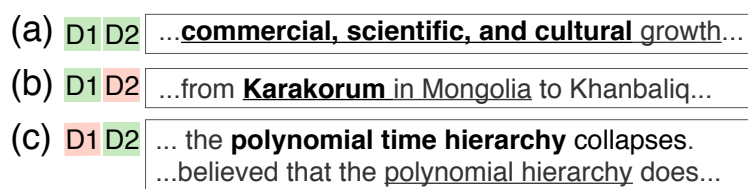


Figure 2.8: Example instances that fall into different user-defined *Boundary* groups.

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.1.4). They also *rejected some prior hypotheses* after using Errudite, such as the distractor case in §2.3. Participants rated their findings to be important ($\mu = 3.7$, $\sigma = 1.12$), were confident that their findings were valid ($\mu = 4.0$, $\sigma = 1.05$), and consistently agreed that Errudite made finding insights easier ($\mu = 4.9$, $\sigma = 0.35$). Participants agreed that they *learned more about the model* ($\mu = 3.9$, $\sigma = 0.94$), and valued Errudite’s support for assessing their hypotheses.

2.5.3 Usability and User Feedback

When rating the usefulness of different components of the tool (Figure 2.9), users rated the DSL ($\mu = 4.8$, $\sigma = 0.40$) and the attribute distribution ($\mu = 4.3$, $\sigma = 0.78$) as very useful, and rated query suggestions ($\mu = 3.6$, $\sigma = 0.91$) and rewrite rules ($\mu = 3.6$, $\sigma = 1.11$) as potentially useful. We hypothesize that rewrite rules pose a learning curve that makes them difficult to evaluate

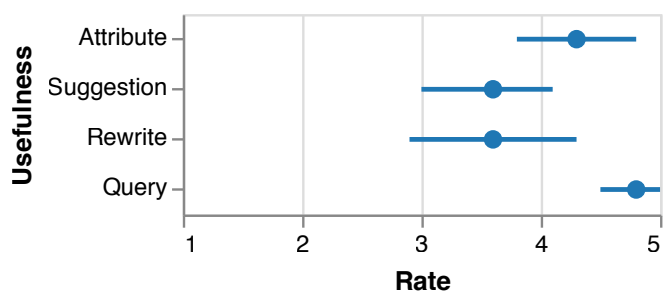


Figure 2.9: Usefulness of function modules in the tool.

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.

2.6 Related Work

2.6.1 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 §2.1, 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., 2019] 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.

2.6.2 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.

2.7 Conclusion and Discussion

In this chapter, 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.1.1 and A.1.2, together with further analysis on SQuAD (Appendix A.1.4). 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.

Going beyond error analysis, the two building blocks of Errudite, *grouping* and *counterfactual rewriting*, are also fundamental to other AI development stages. However, in those cases, relying solely on domain knowledge (as in error analysis) can introduce unnecessary bias. For example, when experts sanity check training data quality, they need groupings to detect conflicting labels between similar instances. Here, handcrafted filtering rules may be subjective and produce unrepresentative groups. In the next two chapters, we further introduce automated algorithmic assistance to mitigate human bias in grouping for data assessment (Chapter 3) and counterfactual data augmentation (Chapter 4).

Chapter 3

AUGMENTING EXPERTS VIA SYSTEMATIC GROUPING FOR ASSESSING DATASET ASSESSMENT

A key challenge for analyzing text queries and assessing the dataset qualities is organizing queries into interpretable, context-preserving, representative, and flexible groups, without inducing human sampling biases. We present *structural templates*, abstract queries that replace tokens with their linguistic feature forms, as a query grouping method. The templates allow analysts to create query groups with structural similarity at different granularities. We introduce *Tempura*, an interactive tool that lets analysts explore a query dataset with structural templates. Tempura summarizes a query dataset by selecting a representative subset of templates to show the query distribution. The tool also helps analysts navigate the template space by suggesting related templates likely to yield further explorations. Our user study shows that Tempura helps analysts examine the distribution of a query dataset, find labeling errors, and discover model error patterns and outliers.

This work was done in collaboration with Kanit Wongsuphasawat, Donghao Ren, Kayur Patel and Chris DuBois during my internship at Apple Inc, and was originally published at CHI 2020 [Wu et al., 2020].

3.1 Introduction

One fundamental functionality in Errudite is to systematically gather and modify instances within groups. In fact, data grouping is also essential for dataset exploration, in which practitioners make sure that their sampled data meet their modeling needs (*e.g.*, having required labels and metadata) and do not have any data quality issues (*e.g.*, conflicting labels on similar instances). However, existing methods for finding data groups are insufficient. Errudite’s manual compilation of filters relies heavily on users’ domain expertise, which can be prone to human errors. On the other

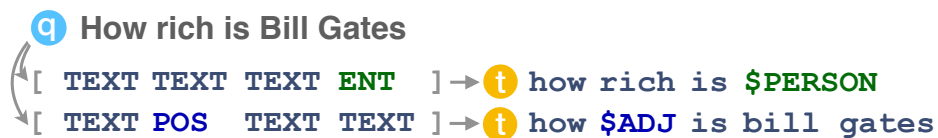


Figure 3.1: Structural templates **t** generated from a given query **q**.

hand, fully automated methods like text clustering can produce groups that are not semantically meaningful. For instance, text clustering [Allahyari et al., 2017] or topic modeling [Chuang et al., 2015] usually produces groups that are difficult to understand. They can hardly distinguish the similarity between “how rich is Bill Gates” with “how rich is Jeff Bezos”, or with “how old is Bill Gates.” Developers often try to rationalize those groups by looking at examples within the group and guessing what the algorithm did. In fact, various existing grouping methods all face certain challenges. We analyze their limitations, and formalize four requirements for a grouping strategy: To inspire actionable analysis, the generated groups should be *interpretable* and *context-preserving*. To enable unbiased analysis, the groups should be *representative* of the dataset. Finally, groups should be *flexible* enough to cope with developers’ varying needs.

While text data are usually tricky, queries are easier to group than longer forms of text (*e.g.*, from articles or books). This is because queries are generally short and often share similar structures. For example, “how rich is Jeff Bezos” and “how rich is Bill Gates” are both one sentence questions. They are also both identically structured “how rich” questions about a person. In this chapter, we aim to leverage this compact structure in text queries to help developers explore query datasets, and create meaningful groups for query analysis.

First, to group queries, we propose an alternative grouping strategy via *structural templates*, which replace query tokens with abstract ones based on linguistic features. For instance, the template “how rich is \$PERSON” abstracts the query “how rich is Bill Gates” in Fig. 3.1 by replacing the token “Bill Gates” with its named entity type, \$PERSON.

Structural templates satisfy the aforementioned requirements. Directly generated from the query dataset, they can *represent* the data distribution. Linguistic features used in their generations help *preserve context*. They also help developers *interpret* and map between templates and the

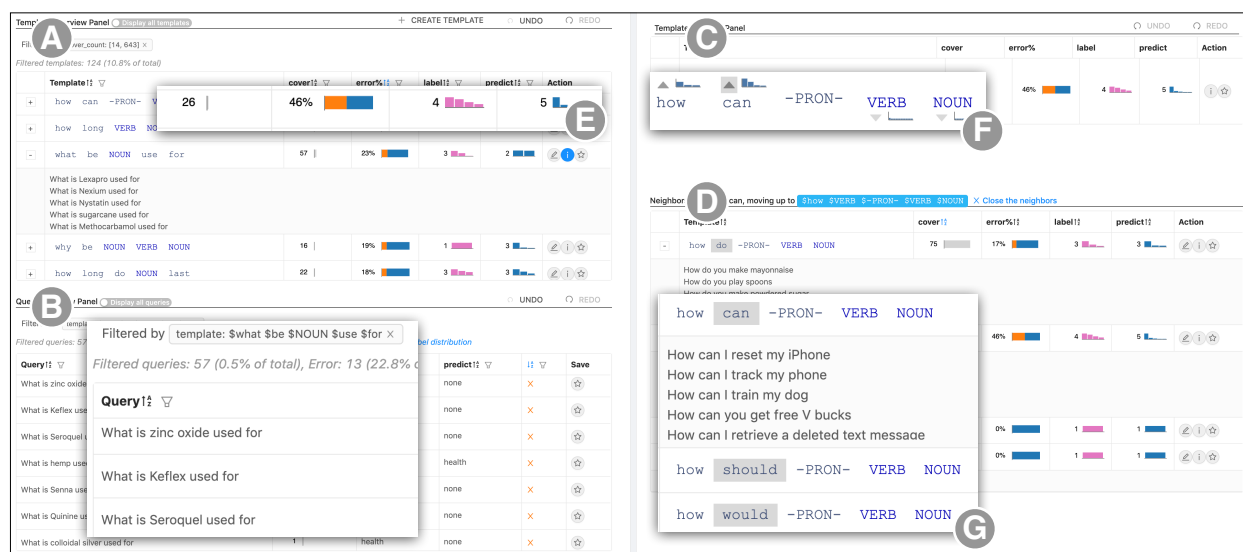


Figure 3.2: Tempura interface. (A) The *template overview panel* lists structural templates automatically selected to represent the dataset’s query distribution. Each row is associated with attributes summarizing their corresponding queries (E, details in Fig. 3.5). (B) The *query panel* lists individual queries in the dataset. (C) The *template traversal panel* allows users to navigate related templates. A focused template selected for traversal (F, enlarged in Fig. 3.8) has visual indicators suggesting related templates to explore. Clicking on the traversal options attached to each linguistic feature opens up the neighbor template table (D). The table displays the corresponding parents or children, with the differing linguistic feature highlighted (G). Like in (G), all templates allow previewing the example queries covered by an expanded template. Video demo: https://youtu.be/s_ODGuZU4G8.

corresponding queries. As we generate many templates for each query using different combinations of linguistic features, analysts can *flexibly* select templates to explore the query dataset from various aspects. For example, instead of replacing “Bill Gates” with $\$PERSON$ to find rich celebrities, one can instead replace “rich” with a part of speech (POS) tag $\$ADJ$ to explore other queries about “Bill Gates” (e.g., “how *old* is Bill Gates”). Analysts can also change the granularity of the grouping. From the template “how rich is $\$PERSON$ ”, one can further abstract $\$PERSON$ into a POS tag $\$NOUN$. The resulting template “how rich is $\$NOUN$ ” will then include previously omitted queries like “how rich is *feta cheese*”, and reveal ambiguities in the word “rich”.

Next, to help developers analyze queries with structural templates, we introduce an interactive system called *Tempura* —**template based query analysis** (Fig. 3.2). With *Tempura*, developers may begin their analyses by looking at the *template overview* panel, which presents a set of templates to summarize the distribution of a query dataset. Alternatively, they may find a particular query of interest in the *query panel*. Via *interactive template traversal*, developers may navigate between query groups with different grouping aspects and granularities. Underlying *Tempura*, we present an entropy-based measure to suggest templates that developers may want to traverse to. Based on this measure, we also develop a *template summarization algorithm* to select a template subset for the overview.

We evaluate *Tempura* both in terms of the template selection algorithm and the system. We demonstrate that the algorithm can reveal distributional differences between three datasets including Natural Questions [Kwiatkowski et al., 2019a], MS MARCO [Nguyen et al., 2016], and internal query logs from a commercial conversational assistant. In a user study, eight experienced machine learning (ML) developers analyzed a query dataset and its modeling performance with *Tempura*. Each of them was able to make around five interesting observations on either the data or the model. These findings included better understandings on dataset distribution, labeling noises, as well as patterns and outliers in the model’s errors — some of which are actionable observations that can help them prioritize their decisions on data cleaning and augmentation. Our users agreed that *Tempura* helped them make more observations with less effort, and suggested additional tasks where the structural template can be useful.

3.2 Understanding Query Analysis And Grouping

We interacted closely with a team of seven ML developers who work on query analysis for six months. We studied their workflows through weekly informal interviews and direct observations. Here, we summarize their two primary query analysis tasks, both requiring coherent query groupings. We then analyze the limitations of existing grouping methods, and identify four requirements for effective grouping.

	Interpretable (G1)	Context- preserv. (G2)	Representative (G3)	Flexible (G4)
Query properties	✓	✗	✓	✗
Manual labeling	✓	✓	✗	✓
Clustering	✗	✓	✓	✗
<i>Structural templates</i>	✓	✓	✓	✓

Table 3.1: Four requirements for effective query groupings (G1-G4). Existing query grouping methods all have limitations. Meanwhile, groups created by our structural template satisfy all the requirements.

3.2.1 Key Tasks: Dataset Exploration and Error Analysis

Developers typically need to perform two primary query analysis tasks. The first task is *dataset exploration* (**T1**), in which they broadly explore their data and perform sanity checks. These sanity checks help developers ensure that their sampled data meet their modeling needs (e.g., not under-representing certain use cases) and do not have any data quality issues (e.g., conflicting labels on similar queries). Data exploration requires comparing the distributions of different query groups (similarly phrased queries, those with the same labels, etc.).

The second task is *error analysis* (**T2**), in which practitioners examine wrong predictions (e.g., from bug reports of deployed systems [Holstein et al., 2019]) to understand their models’ deficiencies. In this case, developers instead usually start with an individual query, and then find if certain errors generalize to systematic error patterns. Although these two tasks have different starting points, in both cases developers need coherent groupings of queries to perform their analyses.

3.2.2 Query Grouping: Existing Methods & Requirements

We analyze limitations of existing query grouping approaches, and identify four requirements for effective grouping methods:

G1 *Interpretable*. Groups should have clear definitions, such that developers can predict what queries are included.

G2 *Context-preserving*. Groups should consider the actual sentence context of the queries, not just coarse properties.

G3 *Representative*. Groups should reflect the dataset distribution, without biasing towards or overlooking queries.

G4 *Flexible*. Groups should help developers assess queries that are similar (G4a) from different *aspects* (“how should the queries be similar”) and at (G4b) different *granularities* (“how similar they should be”).

In practice, developers frequently grouping by *query properties* like query length or query types (e.g., question words). However, such groupings only convey those particular attributes, and thus *preserve too little context* (✗ G2) from the queries to form groups that reveal actionable insights. One developer said “*Knowing my model performs poorly on long queries does not lead to next steps.*” Alternatively, developers would assign query groups based on domain knowledge, either manually or through filtering scripts. Such approach may introduce the developers’ biases and cause the groupings to be *unrepresentative* (✗ G3). One could mistakenly filter and focus on a group of “how rich” questions even when the dataset doesn’t contain many such examples [Wu et al., 2019a].

Clustering algorithms compensate both issues, capturing more sentence meaning than query properties and extracting representative groups directly from the dataset. However, developers only rarely rely on them. They are concerned that clustering algorithms may chain unrelated queries together, making the groups less coherent. For example, “how rich is Jeff Bezos” and “how old is Bill Gates” can be in the same cluster, just because they are both similar to “how rich is Bill Gates.” As a result, developers usually have a hard time *interpreting* the groups (✗ G1). They also cannot *flexibly* inspect how the model respond to “how rich” and “how old” (✗ G4).

3.3 *The Concept of Structural Template*

To develop a grouping method that better satisfies aforementioned requirements, we leverage the fact that queries are typically short and often share similar sentence structure, and propose grouping queries via structural templates. Here, we first describe how we generate structural templates from a query, and how a template can represent a query group. We then explain the hierarchical relationship

between templates, which allows analysts to explore the space of similar queries.

3.3.1 *The Assumption on Query Structures*

Tempura is built on the assumption that queries are properly structured to support part-of-speech tagging. While prior work describes real-world queries as potentially under-specified, terse, and context-dependent [Budzik and Hammond, 2000, Capannini et al., 2011, Hoque et al., 2017, Setlur et al., 2019], our assumption is valid here, as we focus on a subset of real-world queries: *knowledge-seeking questions*. Filtering to a query subset is common in practice for routing queries based on their format, intent, or context. For example, ill-defined queries will trigger hard-coded responses like “Sorry I don’t understand.” As a result, we work on self-contained questions, which are usually cleaner and are in line with the interests of developers we interacted with. The questions are also mostly well-formed in various question answering datasets (e.g., SQuAD [Rajpurkar et al., 2016b], VQA [Antol et al., 2015]) and public query datasets (e.g., Natural Question and MS MARCO similarly discard non-question queries), to which Tempura would directly apply. That said, the syntactic patterns of ill-structured queries are still useful, and we envision how Tempura can be extended for those queries in §3.10.

3.3.2 *Template Generation: Abstracting Query Tokens*

We define a *structural template* to be an abstract query that replaces their original tokens with *abstract tokens*. To produce abstract tokens, we use four linguistic features, listed from the most specific to the most abstract:¹

1. Lower cased original TEXT (“How” → “how”)
2. Normalized word form LEMMA (“is” → “\$be”)
3. Named entity type ENT (“Bill Gates” → “\$PERSON”)
4. Part-of-speech POS (“Bill Gates” → “\$NOUN”)²

¹We color the abstract tokens in the templates based on their linguistic features: `text`, `$lemma`, `$named entity`, or `$POS tag`.

²We merge n-grams entities and noun chunks, such that “Bill Gates” can be treated as `$PERSON` or `$NOUN`, rather

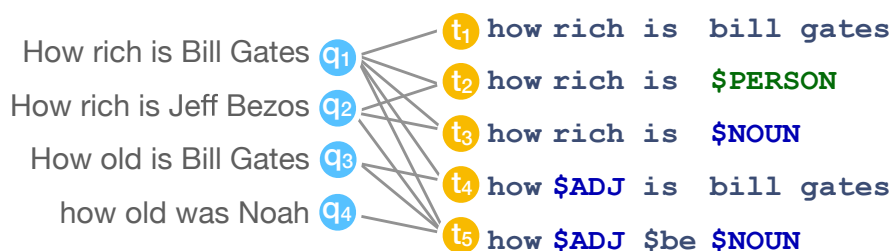


Figure 3.3: Queries and their generated templates form a many-to-many relationship. Each query generates multiple templates, and each template covers a different group of queries.

Each unique combination of the query’s linguistic features produces a template. In Fig. 3.1, the linguistic features [TEXT , TEXT , TEXT , ENT] generate the template that has abstract tokens [“how”, “rich”, “is”, “\$PERSON”]. Because we extract multiple linguistic features from each token, a single query can produce many templates, each with a different set of abstract tokens. For instance, “how rich is Bill Gates” (q_1 in Fig. 3.3) can generate 5 templates. Each template in turn covers multiple queries, and therefore represent a *query group*. In Fig. 3.3, t_2 (“how rich is \$PERSON”) represents a group of two queries (q_1 and q_2).

Templates provide several advantages. First, by pivoting query tokens in a controlled manner, we create template-based groups that are *interpretable* (G1) — analysts can easily map between queries and templates. Second, because templates mirror the syntactic structures of the grouped queries, these groups are *context-preserving* (G2). Finally, as we extract templates from every query in a dataset, the resulting templates can *represent* the dataset distribution (G3).

3.3.3 Hierarchy of Templates with Different Granularity

Because the linguistic features have a ladder of granularity, the templates generated with these linguistic features form an *abstraction hierarchy*. For example, person names and locations are both nouns, so tokens with named entity \$PERSON and \$LOC will all have \$NOUN as the POS tag.³ As a result, the set of queries covered by “how rich is \$PERSON” (t_2 in Fig. 3.3) is always a subset of

than \$PERSON \$PERSON.

³We merge \$NOUN and \$PROPN for simplicity.

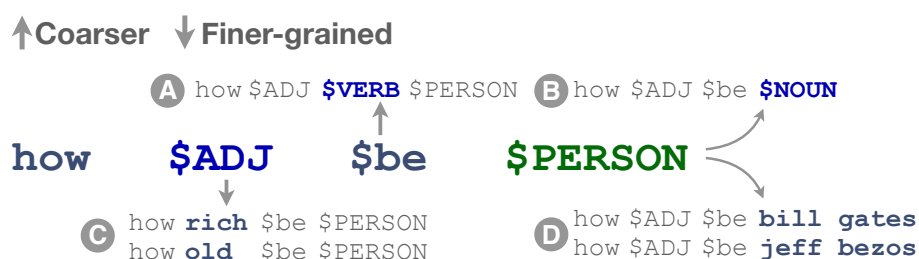


Figure 3.4: Parent and child templates for “how \$ADJ \$be \$PERSON”, by tweaking its different linguistic features: (A) is a parent template found by merging \$be with other possible LEMMA (e.g., \$do) to \$VERB (POS). Similarly, (B) merges \$PERSON with other possible ENT to get \$NOUN (POS). (C) and (D) are two sets of child templates *splitting* (C) the \$ADJ and (D) \$PERSON into actual LEMMA forms.

those in “how rich is \$NOUN” (t_3). In other words, t_3 is an abstraction of t_2 .

The hierarchy provides *flexibility* (G4). Analysts can traverse the hierarchy to examine query groups at different levels of granularity (G4b). By changing a linguistic feature in a template, one can move up to its *parents* (templates that are immediately one level less specific) or down to its *children* (immediately one level more specific). Consider the template “how \$ADJ \$be \$PERSON” in Fig. 3.4: changing the LEMMA “\$be” to its abstract POS tag “\$VERB” generates the parent (A), which includes other \$VERBs besides \$be. Similarly, replacing “\$PERSON” with its abstract POS tag “\$NOUN” produces the parent (B), which covers more general questions by including other \$NOUNs beyond \$PERSON names. On the other hand, splitting “\$ADJ” and “\$PERSON” to their LEMMA forms generate four children, two for (C) and two for (D).

Analysts can also traverse the hierarchy to examine groups from *different aspects* (G4a). For example, from a template that focuses on a specific attribute for different people (“how rich \$be \$PERSON” in Fig. 3.4C), we can pivot to concentrate on different attributes of the same person (“how \$ADJ \$be bill gates” in Fig. 3.4D). To do so, we can first replace “rich” with its abstract POS tag “\$ADJ”, and then replace “\$PERSON” with the TEXT “bill gates”.

3.4 The Tempura Interface

We develop an interactive system, *Tempura*, to help developers use structural templates for query analysis (Fig. 3.2). In this section, we present how the Tempura interface lets developers explore the queries and generated templates by showing overviews first and providing details on demand [Shneiderman, 1996].

3.4.1 Overviews: Bi-directional Starting Point

The Tempura interface provides both template and query overviews. The *template overview* panel (Fig. 3.2A) lists a set of templates to summarize the query distribution as a starting point for dataset exploration (T1). We later describe how Tempura selects templates for this overview in the §3.6 section. The *query* panel (Fig. 3.2B) lists individual queries and lets analysts start error analysis (T2) with one query. To help users locate templates and queries to inspect, both panels support searching with regular expression.

Both panels also display attributes associated with each template and query, and allow users to sort the table entries based on these attributes. The query panel shows a ground truth and a predicted label per query. Meanwhile, the template overview panel presents attributes of each template’s corresponding query group (Fig. 3.2E), including *coverage* (the number of queries covered by the template) and other statistical attributes relevant to the modeling task (explained below).

The template overview is task-agnostic, as the table can show different statistics for different modeling tasks. For example, Fig. 3.5 shows attributes for a text classification task. The stacked bar chart in Fig. 3.5B indicates the model performance with *error rate*, or the ratio of incorrectly predicted queries associated with a template. The distribution bar glyphs in Fig. 3.5C and 3.5D show class distributions for the ground truth and the prediction respectively. In this example, the template contains three ground truth labels, but only two predicted labels (with one dominating). These bar glyphs reveal that the covered queries receive similar predictions, regardless of the actual **\$PERSON** in the query. In contrast, if the template contains only one ground truth label but many predicted labels, the model is likely not robust, and thus can be easily swayed by **\$PERSON**. To see

more details about the distributions, users can click on the glyphs to open a popup window showing an enlarged bar chart (Examples in Fig. 3.9).

3.4.2 Details-on-demand: Multi-aspect/granularity Traversal

After starting their analyses from the two overviews, analysts can further explore query groups of interest on demand with *template traversal*, and thereby get varying views of the data. Basically, analysts can traverse the template hierarchy, by moving up to inspect coarser (more abstract) parents or down to examine finer-grained (more specific) children templates.



Figure 3.5: A template’s attributes, summarizing its corresponding query group: (A) the number of queries covered; (B) the error rate; and the label distribution of queries’ (C) ground truths and (D) predictions.

To explore different query groups with template traversal, analysts can first select a template of interest (Fig. 3.2A). Based on the selection, the traversal panel (Fig. 3.2C) shows a *focused template* on the top (Fig. 3.2F). For each token in the template, the panel presents an arrow and a bar glyph to indicate potential traversal options. The arrow denotes the corresponding token’s traversal direction: moving to children (down) or a parent (up). For example, the arrows in Fig. 3.6 indicate the corresponding two parent and two children sets in Fig. 3.4.

The bar glyphs help preview the templates related to corresponding traversals. For a downward traversal, the bar glyph shows the children templates and their coverage. The four bars in Fig. 3.6C indicates that moving from \$ADJ to its children will generate 4 templates (“how { \$old, \$much, \$big, \$strong } \$be \$PERSON”). We can also see that the coverage of the four children is roughly even. For an upward traversal, the bars represents *sibling* templates (templates with the same parent). The bar glyph in Fig. 3.6B shows that moving from \$PERSON to \$NOUN will merge the current template “how \$ADJ \$be \$PERSON” with another sibling template “how \$ADJ \$be

³For simplicity, we omitted t from the entropy equations (e.g., $S_c(2, \text{LEMMA})$ in C represents $S_c(t, 2, \text{LEMMA})$).

\$LOC”, which has a lower coverage.

When analysts select a traversal direction (by clicking an arrow), Tempura creates a neighbor template table (Fig. 3.2D), which lists all corresponding neighbor templates shown in the bar glyph and their attributes. As children or siblings of the focused template, these templates only differ from each other by one token. Therefore, showing them in one table helps contrast similar query groups.

To help analysts avoid tediously enumerating all the traversal options, we further compute *traversal scores*, and use them to compare and rank all the traversal options. The score measures how informative a traversal action is, or which traversal is likely to yield further exploration. For example, moving from a coarse parent template to its children is informative, when the move provides multiple representative subgroups. We use the color density of the arrow to encode the score. The darker the arrow, the more *preferable* the traversal. In Fig. 3.8C, the dark arrow suggests traversing down **\$ADJ** — an exact example of “provides multiple representative subgroups.” We formalize this score in the next section.

3.5 Scoring Traversals

We introduce the traversal score used to color traversal arrows in the interface. We want to guide the analyst towards templates that have only a few abstract tokens, because they are more interpretable (G1) and context-preserving (G2) than those containing only abstract tokens. However, suggested templates should also cover enough queries to ensure that they are representative enough to be worth exploring.

Suppose an analyst starts with the template t in Fig. 3.6. She can make **\$ADJ** less abstract (C),

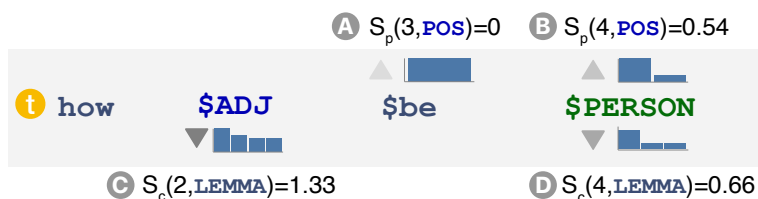


Figure 3.6: Given a focused template t = “how **\$ADJ** \$be **\$PERSON**”, we can traverse to its parents (up arrows on top of **\$be** and **\$PERSON**), or children (down arrow under **\$ADJ** and **\$PERSON**). The color density of the arrows indicates how informative a traversal is. Bar charts denote the coverage distribution of the templates affected by the traverse.³

to inspect all the child templates that have a LEMMA for that token. This action “splits” the current template into finer-grained templates, as in Fig. 3.4C. We encourage this traversal when each of the split children covers a significant number of queries. Conversely, this traversal is not useful when only one or a few children cover the majority of queries associated with t . In such cases, analyzing the high-coverage child yield a slightly more specific subgroup, but at the cost of seeing less queries overall.

To help users weigh this tradeoff, we define a *traversal score* for each available traversal starting from a template t . Let t_i be the abstract token at position i in t , and $|t|$ be the template coverage. Let $C(t, i, l) = \{s \in \text{Children}(t) : s_i = l, |s| > 1\}$ be the subset of children covering more than one query, where we have set the i -th token to be a less abstract token, using a less abstract linguistic feature l . In our previous \$ADJ case, the templates in Fig. 3.4C (also Fig. 3.6C) are examples of $C(t, 2, \text{LEMMA})$. Because all the templates in $C(t, i, l)$ differ on the i -th abstract token, their covered queries are mutually exclusive. For traversing to child templates, $S_c(t, i, l)$, we use the entropy of the normalized coverage of a template’s children,

$$S_c(t, i, l) = - \sum_{s \in C(t, i, l)} \frac{|s|}{|C(t, i, l)|} \log \frac{|s|}{|C(t, i, l)|}$$

where $|C(t, i, l)| = \sum_{s \in C(t, i, l)} |s|$.

We use the entropy to encourage traversals where each of the children have a similar amount of coverage. We discourage traversals when the distribution of coverage among those children is “peaked”; in those cases, the resulting templates have reduced coverage while providing only a small increase to the coherence for each subgroup of queries (G1).

Similar to traversing towards children, analysts can traverse to parent templates, i.e., use a more abstract linguistic feature l for i -th token in t . While traversing towards children splits a template, traversing to a parent conversely merges *sibling templates* (templates that have the same linguistic features as t and have the same parent). For example, templates in any $C(t, i, l)$ are siblings of each other. Just as before, we define a traversal score which encourages traversing to parents whose child templates each cover a significant number of queries. For the parents of t that arise from making

token i more abstract, we compute the same entropy score as before, but instead normalize over the siblings of t , $R(t, i, l) = \{s \in \text{Siblings}(t) : s_i = l\}$. We denote this score $S_p(t, i, l)$.

3.6 Template Summarization

We now present an algorithm that helps select templates for the template overview panel (Fig. 3.2A). Our goal is to select a set of representative templates to summarize the query distribution, and use it as a starting point for dataset exploration (T1).

3.6.1 Intuition: Two Aspects to Consider

To construct representative groups of queries, we prefer templates with high *coverage*. Meanwhile, to effectively initiate the exploration, we want to include templates that are *more preferred* than any of its parents or children. If the parents or children are preferred, analysts will need to perform extra traversing steps to reach the interesting grouping structure.

The *traversal score* defined previously provides a ranking between a given template and all its parents and children. A template t should be selected when we *do not* encourage traversing to any of its parents or children. We define $I(t)$ to be the *maximum* of t 's traversal scores — all the $S_c(t, i, l)$ and $S_p(t, i, l)$ for different tokens i and abstract tokens l :

$$I(t) = \max(\max_{i,l} S_c(t, i, l), \max_{i,l} S_p(t, i, l))$$

Recall that traversal scores represent preferences *towards* parents and children. With $I(t)$ being the maximum, the lower it is, the less likely t should be split or merged at any tokens, and therefore the more likely we want to include it in the overview.

3.6.2 Selection as a Weighted Set Cover Problem

To take these two aspects into consideration, we form the template selection as a weighted set coverage problem. We see the query dataset $Q = \{q_1, \dots, q_n\}$ as the entire set of elements. Then, each template t in $T = \{t_1, \dots, t_m\}$ represents a subset of Q that contains a number of queries $|t|$

(the template *coverage*). We use $I(t)$ as the weights, such that templates with low preferences are penalized by having a high weight. Our goal is to find a set $T^* \subset T$ such that (1) T^* covers at least a user-specified ratio, c , of queries: $|T^*| = |\cup_{t \in T^*} t| \geq c|Q|$; and (2) the sum of the weights of the subsets in T^* is minimized.

Algorithm 1: Template selection

Data: query set Q , generated templates T , min cover. ratio c

Result: a list of overview templates T^*

```

1  $T^* = \{\}$ ;
2 while  $|T^*| < c|Q|$  do
3    $t^* = \arg \min_{t \in T} I(t) / (|T^* \cup \{t\}| - |T^*|)$ ;
4    $T^* = T^* \cup \{t^*\}$ ;
5 return  $T^*$ 

```

Weighted set coverage is a NP-complete problem. Here we use a classic greedy algorithm (Algorithm 1) to compute an approximate T^* [Young, 2008]. The algorithm repeatedly chooses a template t that minimizes the weight $I(t)$ divided by number of queries in t not yet covered by the chosen templates $(|T^* \cup \{t\}| - |T^*|)$. It then stops and returns the chosen templates (T^*) when they form a cover of the original set of queries.

We experiment our selection algorithm on different query datasets. Compared to alternative weighting strategies, our algorithm selects a larger number of templates, but tends to select more interpretable and context-preserving “what \$be \$NOUN” rather than “what \$VERB \$NOUN”). We find the algorithm can reduce the exploration burden: Heuristically, to cover 75% queries of a dataset, the method selects a number of templates that is around 10% of the dataset size (for a 10,000 query dataset, the method selects around 1,000 templates).

3.6.3 Case Study: Templates Selected from Different Datasets

To test whether our automatic template generation and summarization can reveal dataset characteristics, we use Tempura to process three datasets:⁴ (1) Natural Question (NQ) [Kwiatkowski et al.,

⁴We augment linguistic features with SpaCy (<https://spacy.io/>).

2019a] and (2) MS MARCO [Nguyen et al., 2016], which are collections of real queries issued to Google and Bing Search Engines, respectively, as well as (3) anonymized search queries from a commercial conversational assistant (Assistant) that occur on more than 10 different devices. The queries are all seeking general knowledge (not related to personal information). They are automated speech recognition transcripts with no audio attached.

These datasets are used for training question answering systems [Joshi et al., 2017]. We are interested in understanding if queries from different sources differ. To explore this, we sample 10,000 queries from each dataset (with the training and the development set combined) and generate templates from them. We report two major results below.

Crafted research datasets are sparser. The number of generated templates in Table 3.2 reveals NQ’s apparent distributional difference. It generates many more unique templates (around 2.5 million) compared to the other two alternatives (within 1 million). This is likely because additional filters on NQ only keep queries

Dataset	$l(q)$	$ T $	$\# t > 1$
NQ	7.6 ± 1.8	2,662,618	16,976 (0.66%)
MS MARCO	5.2 ± 2.5	992,483	6,601 (0.66%)
Assistant	5.3 ± 1.6	509,734	29,732 (5.83%)

that have (1) more than eight words⁵, and (2) a closely related Wikipedia page. The cartesian product of linguistic features then generates more templates from the longer queries. In other words, while the filters help locate complex questions, they also shift the distribution to be less representative of the natural queries. While MS MARCO generates fewer templates than NQ, it still contains more diverse queries than Assistant: Both NQ and MS MARCO only have 0.66% templates with coverage greater than 1.

Table 3.2: The tested datasets with their query lengths $l(q)$, and the attributes on generated templates. From the total number of generated templates ($|T|$) and the proportion of those covering at least two queries ($|t| > 1$), we see NQ and MS MARCO are more sparse than Assistant.

Real-world queries have different distributions. Further comparing the less sparse MS MARCO and Assistant, we notice that the most representative templates differ. While “**what be**

⁵NQ’s average query length appears to be less than eight in the table, as we merged noun chunks.

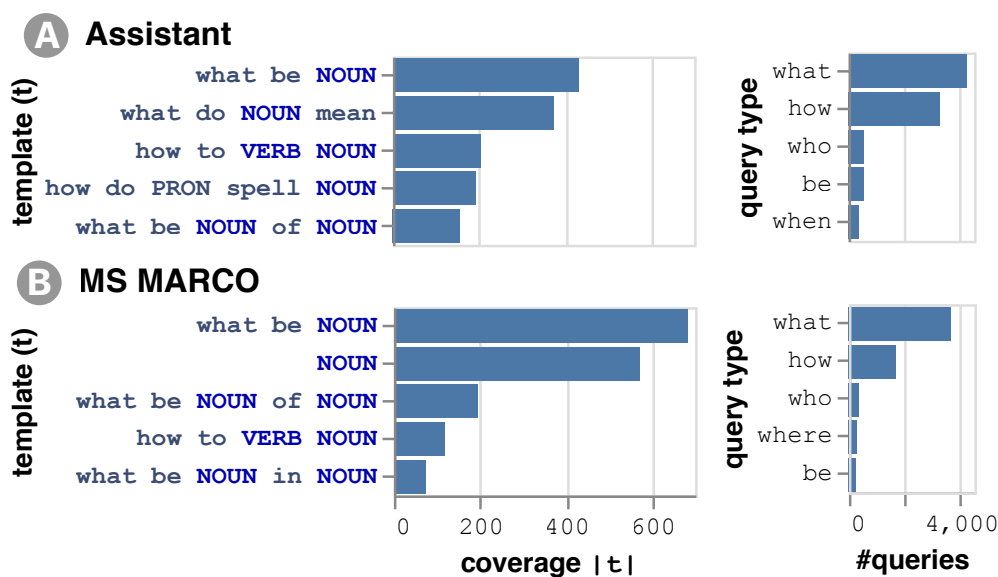


Figure 3.7: The five most representative templates and most frequent query types for the sampled queries in (A) Assistant, and (B) MS MARCO. They have similar query types, but their templates differ after “what \$be \$NOUN”. This indicates that the templates can capture more dataset characteristics than the conventional query types.

“\$NOUN” is an important template in both MS MARCO and Assistant, this template has a much higher coverage in the former. The selected templates diverge afterwards: MS MARCO has many “\$NOUN” (e.g., “401k loan requirements”, “employee turnover types”) while Assistant has more “what \$do \$NOUN mean” queries. The “\$NOUN” case in MS MARCO is especially interesting: all the query datasets were filtered to only keep knowledge-seeking question queries, yet the other two didn’t show any “\$NOUN” queries, indicating different filtering strategies. Such dataset differences are difficult to notice from just the query types (in Fig. 3.7) or query length (in Table 3.2).

3.7 Usage Scenario

We present a scenario to demonstrate how developers can use Tempura to better evaluate a BERT-based classification model [Devlin et al., 2019] on a query dataset. The task is to predict if a query is in one of four categories: *food* (“can I freeze spaghetti”), *health* (“what is Keflex used for”), *procedural* explanations on how to perform some tasks (“how to clean white shoes”), or *none* of

above. The dataset contains 10,000 anonymized, knowledge-seeking queries from Assistant (75% of which are covered by the selected overview templates). It is a sample of the development set that the developers are building a model on. This scenario is inspired by the analyses that participants perform in our user study, which uses the same data.

The developer first filters the templates to only keep those covering more than ten queries, and sorts them by their error rate in descending order (Fig. 3.10). The template with the highest error rate is “how \$can \$-PRON- \$VERB \$NOUN”. Out of the 26 covered queries, 46% are incorrectly predicted. Inspecting them, he notices **noisy labels**, namely similar queries are labeled as asking for a *procedural* explanation (“how can I boil eggs”⁶) and about *food* (“how can I cook salmon”). The developer verifies that queries on cooking procedures make these two label pairs non-exclusive. He treats this observation as supporting evidence to switch to a multi-label classifier.

Afterwards, the developer explores similar templates via template traversal. With the focused template panel (Fig. 3.8), he notices the dark arrow in B, which suggests that merging \$can into its parent POS form (\$VERB) is the most informative traversal.



Figure 3.8: Traversing a focused template. The dark arrow in B suggests moving from \$can to its parent form \$VERB.

Other traversals indeed seem less useful.

The bar charts associated with \$VERB and \$NOUN (C) show that they have a large number of sparse and unrepresentative children. Meanwhile, merging \$how (A) yields a very little gain, as its two neighboring templates have few queries associated with them.

Thus, the developer follows the suggestion, and traverses up from \$can. Doing so triggers the neighboring template panel (Fig. 3.2D) to present sibling templates with different \$VERBs. He notices that the dataset contains a large number of queries in the form of “how \$do \$-PRON- \$VERB \$NOUN”. By expanding the bar charts for the ground truth labels (Fig. 3.9), we see an interesting **label distribution difference** between the two templates: the queries under the \$can

⁶All the queries referred in the chapter have a query frequency larger than 10, i.e., occur on at least ten unique devices.

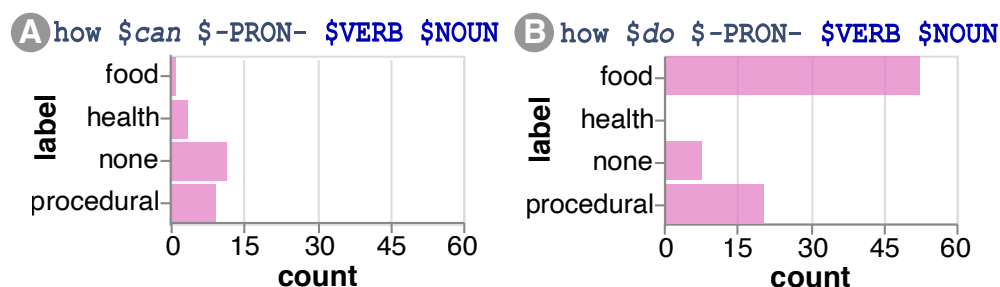


Figure 3.9: The ground truth label distributions for queries covered by two closely related neighboring templates, with (B) “how *\$do* \$-PRON- \$VERB \$NOUN” having a much larger portion of *food* related queries.

template are labeled as *health*, *none*, and *procedural* while those under *\$do* are mainly *food* related. The developer notes that queries in these two forms should be similar, and proposes to augment the data (i.e., to create another batch of food questions by replacing “do” with “can”).

Despite the distribution differences between “\$can” and “\$do”, the developer deems these templates identical, and moves to inspect “how \$VERB \$-PRON- \$VERB \$NOUN”. Afterwards, he instead traverses down the second *\$VERB* to understand what are the actions being queried. Fig. 3.10 shows a neighbor template table for the traversal. Among the four child templates covering most queries, “how \$VERB \$-PRON- \$cook \$NOUN” has the most indicative verb, with all queries labeled and predicted to be *food* related. Meanwhile, “how \$VERB \$-PRON- \$do \$NOUN” has high error rate with only one predicted label (*none*). Retrieving its queries, the developer notices that all but one are labeled as *procedural*. He suspects the model neither understands the template, nor recognizes the *\$NOUNs* (“division”, “a screen recording”, etc.) This **model error pattern** suggests the model has underfitted to this form of query. The lemma *\$make* has the highest coverage. The model predicts most of the “\$make” queries to be *food*. However, 6 of 33 queries are labeled as *procedural* despite involving alcoholic beverages (“how do you make vodka”, “how do you make a margarita”). In hindsight, these queries could arguably have both labels. Furthermore, in *\$play*, we see a **model error outlier**: with “chess” and “old maid” both being board games, “how do you play chess” is labeled as *procedural* but predicted as *none*, whereas “old maid” shows the reverse: labeled as *none* but predicted as *procedural*. Our developer notes the conflict and decides to double

















	Coverage	Error rate	Label	Predict
\$make	34 	21% 	3 	3 
\$cook	10 	0% 	1 	1 
\$play	6 	50% 	2 	2 
\$do	5 	80% 	2 	1 

Figure 3.10: Neighbor template table showing the top four LEMMA (*\$make*, *\$cook*, *\$play*, and *\$do*) after splitting the second *\$VERB* in “how *\$VERB* *\$-PRON-* *\$VERB* *\$NOUN*”.

check the labels, as well as the model stability on various similar *\$NOUN*s.

Hoping to see a comprehensive list of procedural queries, the developer samples more queries with the *procedural* label. He notices many queries asking “how to do (something).” He generalizes from those queries to a template, “how to *\$VERB* *\$NOUN*”. He finds that this template (with 35 queries) has a higher portion of food queries (85.7%), but most queries are semantically similar to those in “how *\$VERB* *\$-PRON-* *\$VERB* *\$NOUN*”. He creates a single, regular expression-related template capturing both subgroups: “how (*\$VERB* *\$-PRON-* |to)+ *\$VERB* *\$NOUN*”, so he can revisit these queries in future investigation.

3.8 User Study

We ran a user study to answer the following three questions:

- Q1 Can Tempura support data exploration and error analysis, and help participants make actionable observations easily?
- Q2 How do analysts decide which templates to inspect?
- Q3 Are Tempura’s components useful for exploration?

Each user study session lasted for one hour. We first surveyed participants’ query analysis experience. Next, we provided a tutorial outlining the features of Tempura. Participants then used Tempura to explore the same data and model as described in §3.7. We encouraged participants to

Observation Type	Count	Examples
Dataset distribution <i>observed frequent query patterns</i>	7 (27%)	“how long \$do \$-PRON- \$VERB \$NOUN (for)*” (148 queries) are all <i>food</i> related (e.g., “how long do you boil corn”), whereas “how long \$do \$-PRON- take for \$NOUN to \$VERB” (16 queries) has half of <i>health</i> queries (e.g., “how long does it take for a piercing to heal”).
Dataset noise <i>discovered labeling issues or illegitimate queries</i>	7 (27%)	2 out of 86 queries in “can dog (have eat)+ \$NOUN” are labeled as <i>health</i> , but the \$NOUNs are all <i>food</i> (“corns”, “rice”). All the 9 queries in “what \$be the benefit of \$NOUN” are predicted to be <i>health</i> , but 4 are labeled as <i>none</i> .
Model error pattern <i>concluded systematic model error categories</i>	10 (38%)	The model cannot understand rare tokens. The \$NOUNs in “what \$be \$NOUN used for” (57 queries) are all medical related, but the model only predicts 28 (49.1%) to be <i>health</i> — possibly affected by their training frequencies. Overfitting to templates. “is \$NOUN bad for \$-PRON-” (5 queries) are all predicted to be <i>health</i> related, even for “is college bad for you” (supposed to be <i>none</i>).
Model error outlier <i>noticed model’s specific behaviors that are not generalizable</i>	2 (8%)	In “can \$-PRON- \$VERB \$NOUN” (29 queries), similar queries are predicted differently: “can I block a contact” is predicted as <i>none</i> , whereas “Can I block unwanted phone calls” is <i>procedural</i> .

Table 3.3: Participants’ example observations on (1) Dataset distribution (they observed frequent query patterns); (2) Dataset noise (they discovered labeling issues or illegitimate queries); (3) Model error pattern (they concluded systematic error categories); (4) Model error outlier (they noticed model’s specific strange behaviors that are not generalizable, or that related queries are mostly correctly predicted).

think aloud and describe their observations while they explored. We noted down their observations throughout the session (Q1), and also logged their clickstreams for analyzing their exploration process (Q2). Afterwards, we confirmed the recorded observations with the participants, and they rated each observation by the ease of discovery, relative to their prior experience working with queries in the wild. Participants further self-assessed how much more they learned about the dataset and the model, and rated the usefulness of different components in Tempura (Q3). All the ratings were collected in the form of five-point Likert scale [Likert, 1932]. Eight ML developers at Apple participated in our study, all with prior query analysis and modeling experience (worked on 3-10 datasets, $\mu = 6.2$). Below, we answer the questions with the study outputs.

3.8.1 Q1: Users made actionable observations in Tempura.

Observations are on datasets and errors.

As mentioned in §3.7, participants noticed both general patterns and outlier behaviors in dataset and error analysis. Their observations can be divided into four categories, as in Table 3.3. On average, subjects reported 5.4 ± 1.83 observations.⁷ In total, participants made 26 unique observations. Participants made around the same number of observations on datasets and on models, indicating Tempura can support both tasks. They consistently rated that it was easier to make observations in Tempura (4.5 ± 0.73).

Observations are actionable.

Five out of eight participants noted that their observations were *actionable*, and proposed several potential next steps. For example, templates helped them improve their labeling tasks. With templates covering interchangeably labeled queries (data noise issue), they would not only clean up existing labels, but use example queries in those template groups to revise the labeling instructions. Templates' label distributions could also guide training data revision. Participants would overcome the template overfitting issue (i.e., queries covered by one template are all predicted the same, regardless of their labels) by collecting data with the same syntax but different label. In contrast, for underfitting templates (i.e., the correlation between a template and a label is not correctly learned), participants would augment the data with semantically neighboring templates. Templates could also help build targeted and challenging validation sets for testing specific syntax phenomena.

Tempura could be better at error analysis.

Participants agreed that they *understood more* about both the dataset (4.4 ± 0.45) and the model (3.9 ± 0.90). Generally, participants believed their analysis on data distribution was more thorough, because various factors that Tempura does not consider — model architecture, transfer learning

⁷We counted participants' self-reported observations. Recall that we confirmed and asked them to rate each one of their observations.

effect — can all account for an imperfect model. They offered suggestions on improving Tempura for error analysis. For example, to prioritize *problematic* templates, we could select overview templates using entropies on model related measurements (e.g., error rates). Such measurement can further surface templates related to error analysis, but we worry it would focus too much on a specific model, at the cost of general dataset information. One participant also mentioned that adding word frequencies could help understand whether the model is overfitting to particular templates or keywords. In fact, structural templates can be paired with many conventional attributes (query length, sentiment, etc.) Further investments are needed to understand which additional debugging attribute provides the most comprehensive view.

3.8.2 Q2: Attributes as the primary clue.

We retrieved participants' clickstreams affecting the overview table, and programmatically labeled them into four types: finding templates that (1) have a certain attributes pattern like high error rate or coverage (*Attr*), (2) contain queries with specific ground truth or prediction label(s) (*Label*), (3) satisfy a partial template search filter, in the form of regular expression pattern (*Regex*), and (4) are generated from a specific query (*Query*).

The results revealed that participants surfaced templates they would like to inspect via all strategies, but the most common ones were by sorting or filtering *Attrs* (used by all participants), or particular *Labels* (seven participants). Users' free form responses reflected that they highly valued the coverage and error rate. Three participants (P3, P6, P8) also commented that the label distributions were useful. P6 said he actively searched for structural overfitting (templates with only one prediction label). P8 expressed particular interest in templates with more varying labels: *"Sometimes we locally improve models for queries with certain patterns. This helps me understand, for example, if the data augmentation on a specific label has negative effects on queries with the same syntax but different labels."* Only four participants generalized template structures from a *Query*. P7 explained that generalization required more of a mental model on the template-query mapping, and thus was harder than starting from template overviews.

3.8.3 Q3: *Tempura is effective; Users learned more about the dataset.*

When assessing the usefulness of different components, users rated the algorithm selected overview templates (4.2 ± 0.63), cross-filtering between templates and queries (4.4 ± 0.68), and supportive attributes (e.g., error rate) & interactions (e.g., sorting) (4.7 ± 0.41) as very useful, and rated template traversal (3.6 ± 0.83) as potentially useful. They especially liked the combination of templates and attribute summaries, saying it greatly helped redirect their attention to important query groups. The lower rating for traversal could be due to the size of the dataset. As P4 pointed out, *“With 10k queries, interesting hierarchical groups are less common. I found myself sometimes getting many low coverage child templates.”*

Participants all thought Tempura was intuitive to learn, and at the same time it greatly enhanced their query exploration experience. Five users commented that Tempura offered a finer-grained and systematic analysis process, making their observations more precise. They mentioned that structural templates provided more efficient and intuitive starting points for categorizing raw queries, helped them to bypass the large coding overhead needed, and sped up their otherwise tedious manual process. The speed up was significant. As P1 described: *“I learned 7 new things in the last half an hour! Without the tool, I would spend all those time writing Python scripts without knowing if it will lead to anything significant.”*

3.9 Related Work

3.9.1 Query Text Analysis

Seeking to reveal systematic and actionable patterns in a dataset, existing analysis methods usually slice text queries into groups-of-interest in various ways. However, grouping unstructured queries is challenging. Groups created manually or via grouping scripts [Wu et al., 2019a, Ratner et al., 2017] are usually not representative of the dataset distribution. Filters on query properties can scale (e.g., word frequency [Felix et al., 2017], query length or answer type [Wadhwa et al., 2018, Kafle and Kanan, 2017]), but prior work has noted that such features usually could either overlook the context of the whole sentence [Mohasseb et al., 2018, Samory and Mitra, 2019], or get too abstractive to be

precise and interpretable [Chuang et al., 2015].

Meanwhile, without isolating features, researchers have also tried to organize the dataset by automatically classifying [Shen et al., 2006, Arguello et al., 2009] or clustering [Allahyari et al., 2017] similar queries together. To date, such algorithms measure similarities with TF-IDF [Allahyari et al., 2017] or embedding-based [Wang et al., 2015] distances, which usually result in query groups with mixed semantic (“when was the colored TV invented” v.s. “when did we invent the colored TV”) and structural similarity (“when was the car invented”). Though mixed groups are noisy, both semantic- and structure- based grouping support unique analysis tasks in isolate: Semantically similar queries can help identify paraphrases, reveal specific contents being queried, or evaluate model robustness on semantics-preserving perturbations [Ribeiro et al., 2018]. Meanwhile, structurally similar ones identify queries with common intents (e.g., queries under “*is \$NOUN good for me*” are all associated with health related issues), and can serve as the basis for query dataset augmentation [Fourney and Dumais, 2016] (e.g., augment *\$NOUN* with additional medicine names in “*is \$NOUN good for me*”, or rewrite all the covered queries to “*what does \$NOUN do to me*”). Our work prioritizes structural similarity, as this features could be more explicitly measured, and thus are more interpretable.

3.9.2 *Template/Pattern-based Analysis*

Structural templates have been implicitly used for various tasks. In question answering [Wadhwa et al., 2018, Shen et al., 2017], analysts inspect queries based on their question types (“what”, “who”), which could be viewed as structures of a short phrase in a query. However, these “templates” are usually too shallow, covering up to three words in their lemma forms. In a more explicit manner, rule/pattern-based methods have been extensively used in information extraction [Hoffmann et al., 2015, McCallum, 2005, Pasca and Van Durme, 2007]. Hearst [Hearst, 1992] identified a set of lexico-syntactic patterns (again, structures of part of the sentence) to recognize hyponyms from free-form text. This approach helps ground various studies on extracting semantic phrases (drug entities from online medical form [Gupta et al., 2014], product features from reviews [Popescu and Etzioni, 2005], etc.) Later, Ratner et al. [Ratner et al., 2017] recognized pattern-based heuristics as

one of the primary source for writing labeling functions in their data programming system, Snorkel. In Errudite [Wu et al., 2019a], Wu et al. similarly allowed linguistic pattern query for grouping instances or doing counterfactual analysis. Mohasseb et al. [Mohasseb et al., 2018] enumerated typical syntactic structures for three different types of web queries, and used such structures as features for query classification. However, these studies require manual compilation of templates, which is tedious and prone to human errors and biases. In contrast, Tempura automatically mines templates from a given dataset. If used as a basis for Snorkel or Errudite, it could compensate the potential biases in analysts’ prior knowledge.

Prior work has also explored automatic template generation. Li [Li, 2010] structured queries’ noun phrases by identifying intent heads (primary objects) and their associated attributes. His templates capture semantic aspects, and tend to include more concrete ontology definitions. While our templates rely on standard POS tags and named entities by default, the preprocessing step can be easily customized and embed more sophisticated linguistic abstractions (e.g., ontologies from knowledge graphs [Ehrlinger and Wöß, 2016]). Hu et al. [Hu et al., 2017] proposed to automatically abstract frequent sentence patterns from social media posts. Similarly, SENPAI [Samory and Mitra, 2019] mined patterns for social computing related measurements — credibility, politeness, and sentiment. Both have slightly different definitions than ours, as they focus on raw token-based templates or frequent subspans of the sentences. On the contrary, Tempura takes advantage of linguistic features, and helps answer more targeted analysis questions from multiple aspects, at various granularities.

3.10 Discussions

We contribute Tempura, an interactive tool that lets analysts explore a query dataset with structural templates. To help analysts navigate the template space, Tempura uses a traversal measure that suggests related templates likely to yield further explorations. To provide an overview, we present a weighted set cover algorithm to select a template subset that represents the dataset’s query distribution. We show that the generated overviews can expose distributional differences between industrial and academic datasets, with the former being more sparse. Our user study shows that

Tempura lets developers use meaningful query groups to investigate modeling issues and improve their models. As the improved models get deployed, we believe Tempura can help enhance the end-user experience.

3.10.1 *Implications beyond Data Exploration and Error Analysis*

Our work has broader implications beyond direct use cases.

First, as a framework, the structural template helps us engage with non-technical ML stakeholders. For example, to enhance model evaluation, *quality analysts* and *designers* can build template-based diagnostic sets to address model biases (e.g., requesting uniform predictions on “\$-PRON- is a \$NOUN”, with \$-PRON-=[“she”, “he”], and \$NOUN=[“doctor”, “nurse”].) To de-noise the data, *dataset requesters* can implement quality controls in the crowdsourced labeling process, and flag queries that are labeled inconsistently from structurally similar ones.

Second, our analysis results encourage future explorations on data understanding and wrangling. The distributional differences between datasets in our case study encourage researchers to design dataset comparison tools, so to help developers assess whether pre-trained models are suitable for a seemingly similar domains. On the other hand, half of the errors in our user study are data related, emphasizing the importance of data qualities. More in-depth studies can be conducted to explore the impact of various data wrangling techniques on model improvement. For example, rectifying distribution gaps between neighboring templates with data augmentation might be effective for fixing annotation artifacts [Gururangan et al., 2018a] (e.g., “how \$do/\$can \$-PRON- \$VERB \$NOUN”) — a scenario we explore in Chapter 4.

3.10.2 *Limitations and Future Extensions*

We discuss the limitations introduced by our design decisions and their corresponding future enhancements.

Assume queries are well-structured. Tempura is currently implemented and tested for well-structured queries. However, the syntactic patterns of ill-structured or incomplete queries (e.g.,

“\$PERSON net worth”, “weather \$CITY”) can still be quite useful to users. In fact, with these queries being shorter and more to the point, we can potentially generate templates with higher coverage. We believe Tempura can handle these queries if we switch to more advanced taggers (e.g., Ganchev et al. [Ganchev et al., 2012] reported 94% tagging accuracy on real-world search logs). On the other hand, complete but long queries could generate templates too sparse to explore. One possible solution is to automatically mine a compact set of partial templates by omitting insignificant structures from queries. The insignificance can be defined by either statistics (similar to frequent pattern mining [Han et al., 2007]), or the parsing tree structure (trimming subclauses on a tree, removing stopwords.)

Prioritize syntactic structures over semantics. As mentioned in §3.9, Tempura primarily focuses on the unique benefits of syntactic similarities. While syntactic structures can capture semantics to some extent, one important future direction is to incorporate *more* semantic understandings. The most straightforward method is to enable more semantically meaningful annotations. For example, with knowledge graphs, tokens like “apple” in [Setlur et al., 2019] can have word sense labels (*fruit* or *company*). Using embedding space similarities, synonyms like “deadliest” and “fatalities” [Hoque et al., 2017] can be grouped beyond \$NOUN. Beyond token-level semantics, we can also enhance sentence-level grouping by merging structural paraphrases into larger semantic groups. Tempura currently achieves such merging via manually created, regular-expression based templates, but more advanced paraphrasing detection models [Yang et al., 2019] can further automate it.

Select template overview with a pre-defined objective. While our traversal score and selection algorithm enable the overview+details user experience, depending on analysts’ objective, alternative methods could be more effective. Like mentioned in §3.8, entropy scores on error rate could help surface structures that a model perform poorly on. While *stable* templates save analysts’ time, doing the reverse and selecting templates with informative traversal options can expose more interactive exploration options. Future work exploring the algorithmic space is needed, such that Tempura can cope with analysts’ different primary objectives.

Chapter 4

AUGMENTING EXPERTS VIA COUNTERFACTUAL REASONING FOR EXPLAINING, EVALUATING, AND IMPROVING MODELS

While counterfactual examples are useful for analysis and training of NLP models, current generation methods either rely on manual labor to create very few counterfactuals, or only instantiate limited types of perturbations such as paraphrases or word substitutions. We present POLYJUICE, a general-purpose counterfactual generator that allows for control over perturbation types and locations, trained by finetuning GPT-2 on multiple datasets of paired sentences. We show that POLYJUICE produces diverse sets of realistic counterfactuals, which in turn are useful in various distinct applications: improving training and evaluation on three different tasks (with around 70% less annotation effort than manual generation), augmenting state-of-the-art explanation techniques, and supporting systematic counterfactual error analysis by revealing behaviors easily missed by human experts.

This work was done in collaboration with Marco Tulio Ribeiro, Dan Weld, Jeffrey Heer, and was originally published at ACL 2021 [Wu et al., 2021].

4.1 Introduction

In Chapter 2, we established that counterfactual analysis and reasoning — mentally simulating what *would have happened* if conditions were different — is a common tool for making causality assessments [Kahneman and Tversky, 1981], which in turn are crucial for model evaluation, error analysis, and explanation [Miller, 2019]. For example, in Figure 4.1, “It is great for kids” is perturbed into multiple variations, each providing unique insights by simulating what would have happened if the sentence was different.

Applications of counterfactual reasoning to NLP generally specify the relationship $x \rightarrow \hat{x}$, and

then create \hat{x} according to the relationship. As a result, prior work has tailored counterfactual generators for different applications, only collecting subsets of \hat{x} that are useful for the specific task. For example, to support *model training and evaluation*, human annotators create counterfactuals that change the groundtruth labels by manually rewriting instances [Gardner et al., 2020, Qin et al., 2019] or defining perturbation functions [Ribeiro et al., 2020]. Manual rewrites are costly (*e.g.*, 4–5 minutes per counterfactual [Kaushik et al., 2020]) and susceptible to systematic omissions (*e.g.*, human annotators may cover **great** \rightarrow **not great**, but miss **kids** \rightarrow **no one** in Figure 4.1B). Meanwhile, automated generators for *model analysis and explanation* usually focus on other relationships, *e.g.*, generating \hat{x} that have different model predictions than x [Ross et al., 2021, Zhang et al., 2019a]. As a result, they neglect prediction-preserving counterfactuals that are equally important for understanding or shaping model behaviors, like **kids** \rightarrow **no one** and **great** \rightarrow **scary** linked to Figure 4.1D.

However, counterfactual generation does not *have* to be task-specific. The same set of counterfactuals in Figure 4.1 can support a variety of applications. Moreover, for cases like model explanation and analysis, a general-purpose pool of counterfactuals may be preferable, as the relationship of interest can be more exploratory and user-oriented [Wu et al., 2019a]. In this work, we formalize the task of *counterfactual generation*, disentangling generation from the application of counterfactuals. Given an input x (Figure 4.1A), our generator produces a set of counterfactuals $\hat{\mathbf{X}} = \{\hat{x}_1, \hat{x}_2, \dots\}$ with *application-agnostic*

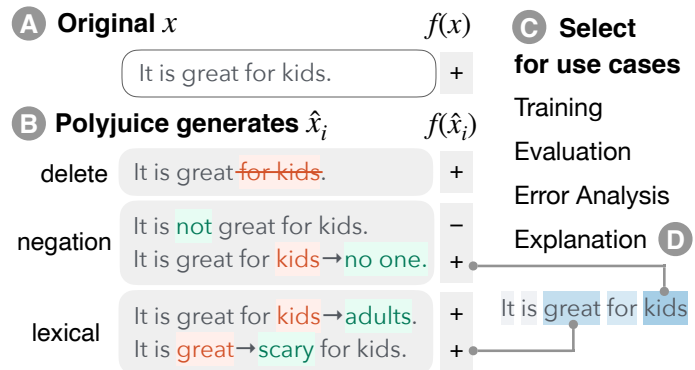


Figure 4.1: Overview: (A) given a sentiment analysis instance x , POLYJUICE¹ generates (B) various counterfactuals \hat{x} , which are then (C) selected for downstream use. *e.g.*, in (D) we select counterfactual explanations that complement a black box explanation: though “great” and “kids” are deemed important, perturbing them may not affect the prediction $f(x) = f(\hat{x}) = \text{positive}$, revealing model failures not covered by feature attributions.

¹We open source POLYJUICE at <https://github.com/tongshuangwu/polyjuice>.

relationships $x \rightarrow \hat{x}_i$ (Figure 4.1B). Afterwards, we use *application-specific* selection methods to find subsets of \hat{x} that are most effective for a given use case (Figure 4.1C).

We frame the generation step as conditional text generation, and finetune GPT-2 [Radford et al., 2019] into a generator called POLYJUICE using (x, \hat{x}) pairs. To allow for targeted counterfactuals, we also design control codes like `negation` or `delete` (Figure 4.1B), and adopt fill-in-the-blank structures [Donahue et al., 2020] to specify where the perturbation occurs and how. Intrinsic evaluation shows that POLYJUICE generates \hat{x} that are *fluent*, *diverse*, and *close to x*, and that the *control* mechanisms retrieve perturbations that would likely not be sampled from off-the-shelf language models.

With simple selection heuristics, we show that a single POLYJUICE model can significantly aid humans in diverse downstream applications.² For *counterfactual training and evaluation* (§4.3), humans label POLYJUICE counterfactuals rather than creating them from scratch. They produce training data that significantly improve model generalization, as well as contrast sets that help identify model vulnerabilities [Gardner et al., 2020], with around 70% less annotation effort. In another application, POLYJUICE produces *counterfactual explanations* (§4.4), providing significant insight on top of state-of-the-art explanation techniques. Finally, POLYJUICE supports counterfactual *error analysis* (§4.5). It allows users to explore related counterfactuals (*e.g.*, the model responds differently to different negation forms in Figure 4.1B), and to aggregate individual counterfactuals into patterns in order to gain systematic understanding of model behavior.

4.2 General-Purpose Counterfactuals

4.2.1 Definition and Desiderata

Given an instance x , a generator g produces a set of counterfactuals $\hat{\mathbf{X}} = \{\hat{x}_1, \hat{x}_2, \dots\}$ with various relationships $x \rightarrow \hat{x}_i$. For example, `great` \rightarrow `not great`, `kids` \rightarrow `no one` in Figure 4.1B are both instances of the `negation` relationship. Each (x, \hat{x}) pair shares multiple relationships — these two are also

²We demonstrate POLYJUICE in semi-automatic settings, but as discussed in §4.2.2, it can also work automatically.

instances of the *label flipping* relationship if the task is sentiment analysis (but might not be for other tasks). As illustrated in §4.1, knowing which relationships apply aids selection for downstream applications.

We expect g to produce counterfactuals \hat{x} that are (1) **close** to x , preferably only involving the minimal changes necessary to establish a certain effect [Pearl, 2018], allowing users to make causality assessments. The generated \hat{x} should also be (2) **fluent**, *i.e.*, grammatically correct [Morris et al., 2020] and semantically meaningful (*e.g.*, “Colorless green ideas sleep furiously” is not meaningful [Chomsky, 2002]). Fluency operationalizes “probable” counterfactuals in the context of NLP; as Kahneman and Tversky [1981] stated, humans strongly favor counterfactuals that are close to the original instance, but also prefer those that could have easily happened without assuming rare events or strange coincidences. Further, as a general-purpose generator, g should produce counterfactuals with a measure of (3) **control** over relationships $x \rightarrow \hat{x}$, such that the counterfactuals can vary with the object-of-attention in each application (the “focus rule” [Kahneman and Tversky, 1981]). Finally, we expect g to output a (4) **diverse** set of \hat{x} in terms of relationships, covering a large variety of “what-ifs” for different applications [Pearl, 2018].

```

1 It is great for kids.  $x$  <|perturb|>
2 [negation] code
3 A It is [BLANK] great for [BLANK]. [SEP]
4 not [ANSWER] children [ANSWER]  $\hat{x}$ 
5 <|endoftext|>
6 B It [BLANK] great [BLANK]. [SEP]
7 is not [ANSWER] for children [ANSWER]  $\hat{x}$ 
8 [BLANK] [SEP]
9 It is not great for children. [ANSWER]  $\hat{x}$ 

```

Figure 4.2: (A) POLYJUICE prompt format, which concatenates the original x , the control code, and the \hat{x} (“It is not great for children” converted to an infilling structure). At *generation* time, POLYJUICE accepts prompts that just include x (Line 1), or optionally with the code and the [BLANK]s (Lines 2–3), and fills in the blanks sequentially with spans separated by [ANSWER]s (Line 4). (B) POLYJUICE allows blanking at different granularities (even the entire sentence), such that Lines 3–4 in (A) can be replaced by Lines 6–7 or 8–9.

4.2.2 Conditional Counterfactual Generation

We frame counterfactual generation as a conditional text generation task using language models (LMs), and train POLYJUICE by finetuning GPT-2 [Radford et al., 2019] using the following prompt design (alternative LMs could also have been used).

Prompt format design. To ensure that \hat{x} is *close* to x rather than arbitrary text, we condition the generation on x , followed by a special token (Line 1 in Figure 4.2A). In Line 2, we have *control codes* [Keskar et al., 2019] such as *negation*. We design them to specify types of perturbation from among lexical, syntactic, or semantic aspects (see Table 4.1), inspired by prior work that categorizes manually created counterfactuals [Kaushik et al., 2020, Gardner et al., 2020]. As an additional layer of control over $x \rightarrow \hat{x}$, we allow users to specify *where* changes happen by having the LM infill [BLANK] tokens [Donahue et al., 2020], rather than generating arbitrary counterfactuals (Lines 3–4).

Finetuning GPT-2 — a causal LM for predicting next tokens — additionally allows us to exercise control at various levels of granularity. At generation time, if the user provides only the original example, POLYJUICE will generate the control code, the blank locations, and the infilling (Lines 2–4). Alternatively, the user can specify the control code, or the control code *and* the blanks, to exercise different degrees of control depending on the application. As later shown in §4.4 and §4.5, such control is important for different use cases.

Training data. To train a conditional model, we combine six existing sentence-pair datasets, each containing a subset of the desired phenomena in Table 4.1. Further, we find naturally occurring sentence pairs (filtered by edit distance to guarantee closeness) in non-paired datasets including CommonGen [Lin et al., 2020], Natural Questions [Kwiatkowski et al., 2019b], and SQuAD [Rajpurkar et al., 2016a], such that the resulting dataset contains *diverse* counterfactuals.³

We translate these sentence pairs into the format given in Figure 4.2A. For each (x, \hat{x}) , we compute its primary control code using part-of-speech tags and dependency trees. For example,

³We exclude data related to our applications, *e.g.*, PAWS-QQP [Zhang et al., 2019b].

Ctrl code	Definitions and POLYJUICE-generated Examples	Training Datasets
negation	A dog is not embraced by the woman.	[Kaushik et al., 2020]
quantifier	A dog is → Three dogs are embraced by the woman.	[Gardner et al., 2020]
shuffle	To move (or swap) key phrases or entities around the sentence. A dog → woman is embraced by the woman → dog .	[Zhang et al., 2019b]
lexical	To change one word or noun chunk without altering the POS tags. A dog is embraced → attacked by the woman.	[Sakaguchi et al., 2020]
resemantic	To replace short phrases without altering the remaining dep. tree. A dog is embraced by the woman → wrapped in a blanket .	[Wieting and Gimpel, 2018]
insert	To add short phrases without altering the remaining dep. tree. A dog is embraced by the little woman.	[McCoy et al., 2019]
delete	To remove short phrases without altering the remaining dep. tree. A dog is embraced by the woman .	[McCoy et al., 2019]
restructure	To alter the dep. tree structure, e.g., changing from passive to active. A dog is embraced by → hugging the woman.	[Wieting and Gimpel, 2018]

Table 4.1: We design a list of control codes to guide generation. We show POLYJUICE-generated counterfactual examples, and the representative training datasets for each corresponding pattern. Details are in Appendix B.1.

negation occurs when we observe changes to negation modifiers or specific words like “supposedly”, and **shuffle** occurs when we have overlap between tokens deleted and added. When multiple changes occur, we label it with the control code which most significantly changes the semantics of the corresponding subphrase as computed by SBERT [Reimers and Gurevych, 2019]. For example, in Figure 4.2A, **negation** (**great** **→ not great**) is more significant than **lexical** (**kids** **→ children**). To balance the distribution (Table B.1 in Appendix B.1), for each dataset, we extract control codes from all the (x, \hat{x}) ,⁴ and randomly sample up to 10,000 instances per codes.

In order to allow for flexible blanking at generation time, we generate multiple training prompts per pair, covering different dependency tree structures related to the perturbed spans (Figure 4.2B), including (1) just the changed tokens, (2) the associated parsing structures, (3) the merged changes, and (4) the entire sentence. We eventually obtain 657,144 prompts from 186,451 pairs.

⁴We use sentences in a pair interchangeably as x and \hat{x} to learn the control codes both ways.

Fluency filtering. While the original GPT-2 produces *fluent* text, some combinations of control codes and blanks cause POLYJUICE to generate nonsensical results. Following Morris et al. [2020], we score both x and \hat{x} with GPT-2, and filter \hat{x} when the log-probability (on the full sentence or the perturbed chunks) decreases by more than 10 points relative to x . Fully automated uses of POLYJUICE (e.g., adversarial attacks) may benefit from stricter constraints, at the cost of diversity (as surprising changes may be filtered even if they are fluent).

4.2.3 Intrinsic Evaluation

We evaluate POLYJUICE on *closeness* and *diversity* by comparing its perturbations on 300 randomly selected sentences with baselines that use more or less context from x : (1) non-finetuned GPT-2, (2) token-infilling RoBERTa [Liu et al., 2019] and (3) span-infilling T5 [Raffel et al., 2020].

As shown in Table 4.2, POLYJUICE generates counterfactuals that are close to the original instance, measured by syntac-

tic tree [Zhang and Shasha, 1989] and Levenshtein edit distance [Levenshtein, 1966]. In contrast, non-finetuned GPT-2 generates arbitrary text instead of perturbations when given the starting tokens of a sentence, as it only leverages context in a single direction. As for infilling models, POLYJUICE counterfactuals are more diverse (measured by self-BLEU [Zhu et al., 2018]) than RoBERTa ones, which is restricted to word substitution. Meanwhile, T5 displays higher diversity but less closeness, probably due to the fact that it does not consider the original masked tokens when generating \hat{x} . For example, in Figure 4.1 “It is great for kids,” T5 replaces “for kids” with “idea”, “to meet you,” whereas POLYJUICE generates “for kids yet adults can enjoy,” “for any audience.”

We evaluate *controllability* by comparing POLYJUICE with T5 as well as with GPT-2 finetuned on prompts *without* codes. We verify that the codes improve the success rate of generating

Model	Diversity	Closeness	
	Self-BLEU ↓	Levenshtein ↓	Syntactic ↓
POLYJUICE	0.34	0.25	2.13
GPT-2	0.18	0.70	6.35
T5	0.12	9.52	3.50
RoBERTa	0.47	0.14	1.32

Table 4.2: Intrinsic evaluations: POLYJUICE counterfactuals are *closer* to the original instance than non-finetuned GPT-2 and T5, and more *diverse* than RoBERTa. Computational details are in Appendix B.1.2.

counterfactuals with the desired perturbation types set out in Table 4.1 by as much as 42% for perturbations such as *negation* and *insert*. For example, given “It is [BLANK] great for kids,” baselines generate “also,” “fun and,” rather than “not” (*negation*).

We further verify the *fluency* for POLYJUICE counterfactuals in three tasks/datasets: (1) *Sentiment Analysis*, SST-2 [Socher et al., 2013], (2) *Natural Language Inference (NLI)*, SNLI [Bowman et al., 2015], and (3) *Duplicate Question Detection (QQP)* [Wang et al., 2018a]. We randomly select 100 sentences per dataset, generate 3 \hat{x} per x , and ask crowd workers to rate whether they are “*likely written by native speakers*.” The workers rated most counterfactuals as fluent: 78% in SST-2, 76% in QQP, and 86% in SNLI. In subsequent sections, we show these rates are suitable for applications where people “team up” with POLYJUICE.

4.3 Counterfactual Evaluation & Training

We ask crowdworkers to label POLYJUICE-generated counterfactuals for *Sentiment*, *NLI*, and *QQP*, for the purposes of evaluation and training.⁵ In each labeling round, the worker is presented with an original x and its label, and asked to annotate the groundtruth for three \hat{x} , rejecting non-fluent ones (details and interface in Appendix B.2.1).

We use a simple heuristic to select which counterfactuals are presented for labeling, aimed at increasing diversity. Representing each \hat{x} by its token changes, control code, and dependency tree structure, we greedily select the ones that are least similar to those already selected for labeling. This avoids redundancy in the labeling set, *e.g.*, common perturbation patterns such as **black** \rightarrow **white**.

4.3.1 Evaluation with Contrast Sets

We verify whether POLYJUICE counterfactuals can be used to create *contrast sets* [Gardner et al., 2020], *i.e.*, evaluation sets where each instance has a nearby counterfactual with a *different* groundtruth, to better evaluate model decision boundaries. We construct these sets by simply

⁵We collect *asymmetric counterfactuals* [Garg et al., 2019] by sampling more *Duplicate* and *Entailment* examples in *QQP* and *NLI* to perturb, due to the difficulty of flipping other labels.

Task	Dev.	Orig. set	Contrast set ↓	Consistency ↓
<i>Sentiment</i>	94.3	93.8	84.9 (-8.9)	76.1
<i>NLI</i>	86.5	91.6	72.3 (-19.3)	56.4
<i>QQP</i>	91.7	87.5	75.3 (-12.2)	61.1

Table 4.3: POLYJUICE \hat{x} as contrasts sets, with model accuracy on the development set, the original set of x , the contrast sets, and consistency (cases where the model predicts both x and \hat{x} correctly). The performance drops are similar to that of expert-created sets [Gardner et al., 2020], on which the accuracy of all classification models decreases by 9.8 on average, with a consistency of ≈ 64.1 . This indicates POLYJUICE can be used to create such sets without expert annotators and at less cost.

filtering out counterfactuals that are labeled the same as their original instances (40%–63% depending on the task).

For each task, we test multiple classifiers open-sourced by Huggingface [Wolf et al., 2020], and report the best performing model for each⁶ in Table 4.3 (results for other models are analogous). POLYJUICE contrast sets display performance gaps consistent with those of Gardner et al. [2020], where the sets are constructed manually by NLP researchers, even though we use non-expert annotators who only *label* examples rather than creating them.

4.3.2 Training with Counterfactuals

Following Kaushik et al. [2020], we augment training sets with counterfactual examples. In all experiments, we finetune `roberta-base` on datasets of n original examples and m counterfactuals, which are generated by POLYJUICE (`m-polyjuice`) or crafted from scratch by humans (`m-CAD` from Kaushik et al. [2020], only available for *NLI*). To distinguish the benefit of counterfactuals from that of just adding more data, we further add a baseline that uses $n + m$ original examples (`m-baseline`). In addition to in-domain test set accuracy, we measure models’ generalization on out-of-domain datasets, as well as contrast sets and challenge sets. We also evaluate model capabilities with CheckList [Ribeiro et al., 2020] for *Sentiment* and *QQP*. Reported model performances are averaged across multiple data samples and random seeds (Appendix B.2.2).

⁶huggingface.co/roberta-large-mnli, [textattack/roberta-base-SST-2](https://textattack.com/roberta-base-SST-2), [ji-xin/roberta_base-QQP-two_stage](https://github.com/ji-xin/roberta_base-QQP-two_stage)

Model	SST-2	Senti140	SemEval	Amzbook	Yelp	IMDB	IMDB-Cont.	IMDB-CAD
m-baseline	92.9±0.2	88.9±0.3	84.8±0.5	85.1±0.4	90.0±0.3	90.8±0.5	92.2±0.6	86.5±0.2
m-polyjuice	92.7±0.2	90.7±0.4	86.4±0.1	85.6±0.8	90.1±0.0	90.6±0.3	94.0±0.3	89.7±0.5

Table 4.4: *Sentiment* model performance, with $n=4,000$ and $m=2,000$. **Bolded** cells highlight significant improvements. m-polyjuice maintains the in-domain and out-of-domain accuracies on reviews (SST-2, Amzbook, Yelp, IMDb Movie Review [Ni et al., 2019, Asghar, 2016, Maas et al., 2011]), improving it on Twitter data (Senti140 and SemEval 2017 [Go et al., 2009, Nakov et al., 2013]) and contrast sets [Gardner et al., 2020, Kaushik et al., 2020], likely because their distributions are less similar to the original SST-2 training data.

Model	SNLI	MNLI-m	MNLI-mm	SNLI-CAD	break	DNC	stress	diagnostic
m-baseline	85.7±0.4	86.1±0.2	86.6±0.2	72.8±0.3	86.4±1.5	54.5±0.6	65.1±0.6	56.0±0.8
m-CAD	85.8±0.6	86.6±0.1	85.6±0.3	73.8±0.2	89.4±2.9	55.8±0.9	65.5±0.5	56.4±0.4
m-polyjuice	85.3±0.3	86.0±0.1	86.4±0.0	73.6±0.2	89.1±1.2	57.7±0.3	65.1±0.2	57.5±0.5

Table 4.5: *NLI* models, with $n=20,000$ and $m=1,574$. m-polyjuice improves accuracy on contrast and challenge sets [Kim et al., 2019, Naik et al., 2018, Glockner et al., 2018, Wang et al., 2018a]; it exhibits comparable (or better) gains than m-CAD (manual counterfactuals) with less implementation and annotation effort.

For *Sentiment*, we select random POLYJUICE counterfactuals regardless of their labels, as long as an original x has at least one \hat{x} that flips the label. For *NLI* and *QQP*, we observed in a pilot study that randomly chosen counterfactuals may not be more effective than the same amount of additional data. We suspect that POLYJUICE lacks domain knowledge and context for identifying critical perturbations, and therefore brings benefits redundant with pre-training [Longpre et al., 2020]. Thus, we use the slicing functions of Chen et al. [2019] to find patterns of interest (*e.g.*, prepositions in *NLI*), and perturb those patterns by placing [BLANK]s on the matched spans. For example, “His surfboard is beneath him” becomes “His surfboard is [BLANK] him”, and POLYJUICE generates counterfactuals such as “His surfboard is **beneath** \rightarrow **next to** him.”

Results. Tables 4.4–4.6 indicate that POLYJUICE augmentation is effective in all tasks: m-polyjuice maintains in-domain accuracy while *consistently* improving or maintaining generalization accuracy in various out-of-domain and challenge sets. On *NLI*, POLYJUICE counterfactuals are as effective or more effective than counterfactuals created from scratch (m-CAD). Notably, we

obtain the largest gains on challenge and contrast sets (*e.g.*, Break and DNC in Table 4.5) or when the out-of-domain dataset is sufficiently different from the training domain (*e.g.*, Senti140 and SemEval in Table 4.4). POLYJUICE also improves results on CheckList tests that previously had high error rates: it significantly lowers the error rates on 11 out of 27 QQP tests,⁷ making 2/27 tests worse. For *Sentiment*, it improves the model on 5 out of 15 tests, hurting 1. Here, we only report a low m/n ratio (<10% for NLI and QQP) to show that a small amount of augmentation is already beneficial. The results are similar for other combinations we explored (see Appendix B.2.2), except when the ratio of counterfactual to original data was too high (*e.g.*, $m = n$ may decrease vocabulary diversity or induce additional data bias, echoing [Khashabi et al., 2020]).

4.3.3 Discussion

We show that POLYJUICE counterfactuals are useful for evaluation, and more effective than additional (non-counterfactual) data for training in a variety of tasks. In contrast to prior work where humans generate counterfactuals from scratch, we only ask them to *label* automatically generated ones, while still achieving similar or better results.

We believe our approach is more effective than manual creation (although both are beneficial): in terms of implementation effort, the process of just labeling counterfactuals is the same as labeling original examples, such that no additional annotator training or separate pipelines are required; in contrast, Kaushik et al. [2020] set up two separate crowdsourcing tasks for creating and labeling the counterfactuals. Further, annotator effort is much lower, as evaluating examples is easier than creating them — Kaushik et al. [2020] report an average of ≈ 2 minutes per NLI counterfactual *prior* to quality validation, while our median time was 10 seconds per counterfactual. Even after our quality validation (removing noisy annotators, disregarding non-fluent counterfactuals), our rate

Model	QQP	PAWS-QQP
m-baseline	84.5 ± 0.6	37.0 ± 0.5
m-polyjuice	84.7 ± 1.0	38.7 ± 0.4

Table 4.6: POLYJUICE with $n=20,000$ and $m=1,911$ improves accuracy on PAWS-QQP [Zhang et al., 2019b].

⁷The absolute error rate drops for at least 5 points, with a relative difference of more than 10%.

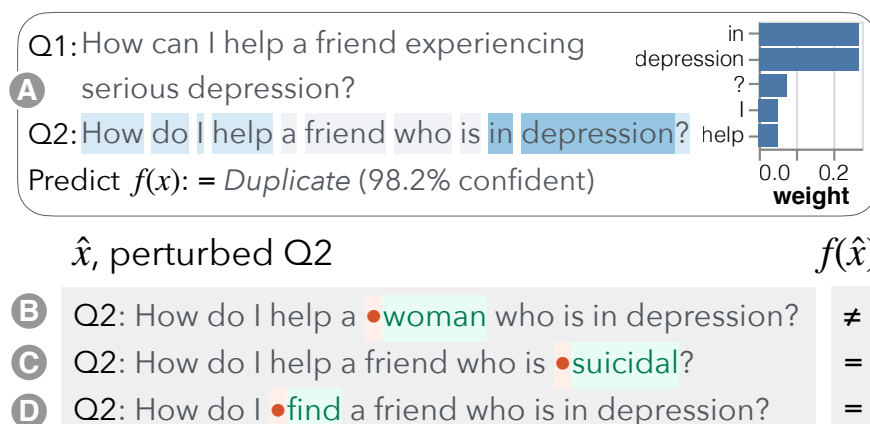


Figure 4.3: (A) An instance in *QQP* where the model prediction $f(x)$ is *Duplicate* (=) at 98.2% confidence, with SHAP importance weights for tokens in Q2. Counterfactual explanations complement SHAP with concrete examples and surprising behaviors, *e.g.*, (B) shows that *friend* → *woman* surprisingly flips the prediction to *Non-Duplicate* (≠), despite the low weight on “friend.”

for *NLI* is ≈ 36 seconds per counterfactual (used in Table 4.5).

In terms of the utility per counterfactual, manual creation and POLYJUICE may be complementary. Manual annotation may be unreliable or incomplete for certain forms of counterfactuals [Ribeiro et al., 2018], whereas POLYJUICE can miss more complex or context-dependent changes, and could benefit from target perturbations that compensate for its lack of domain knowledge (targeted guidance is also helpful for human annotators [Huang et al., 2020b]). Thus, it may be important to mix both approaches [Khashabi et al., 2020]. POLYJUICE’s flexibility opens up possibilities for hybrids between human creation and human verification of targeted, machine-generated counterfactuals.

4.4 Counterfactual Explanations

A popular way of explaining NLP models is to attribute importance weights to the input tokens, either using attention scores [Wiegrefe and Pinter, 2019] or by summarizing the model behavior on perturbed instances (*e.g.*, LIME [Ribeiro et al., 2016] and SHAP [Lundberg and Lee, 2017]). Though ubiquitous, token scores may not always reflect their real importance [Pruthi et al., 2020]. Popular

packages like LIME or SHAP estimate scores by *masking* words, and therefore may not reflect model behavior on natural counterfactual cases. For example, the token “friend” in Figure 4.3A is not considered important even though a natural substitution in Figure 4.3B flips the prediction. The opposite happens to “in depression,” where a significant change makes no difference to the model’s prediction (Figure 4.3C). Even perfect importance scores may be too abstract for users to gain real understanding [Miller, 2019], *e.g.*, users may not grasp the significance of a low importance score for the token “help” without concrete examples such as the one in Figure 4.3D.

Since presenting a large number of concrete counterfactuals would be overwhelming, we propose a hybrid approach, displaying feature attributions as a high-level summary, together with a judicious selection of POLYJUICE counterfactuals that make behaviors concrete and highlight potential limitations. Following Miller [2019]’s observation that people look for explanations revealing *unexpected* behavior, we select *surprising* counterfactuals.⁸ That is, we estimate the expected change in prediction with feature attributions, and select counterfactuals that violate these expectations, *i.e.*, examples where the *real* change in prediction is large even though importance scores are low (Figure 4.3B), and examples where the change is small but importance scores are high (Figure 4.3C). Of course, users can also view additional counterfactuals that perturb tokens of particular interest, a technique that we explore in the next section.

User evaluation. We study the scenario where an expert has access to a model and local explanations, and evaluate the *additional* benefit of showing counterfactuals, *i.e.*, whether they bring *new* insights. We evaluate three ways of generating counterfactuals: (1) POLYJUICE-*random*, a baseline where we show random POLYJUICE counterfactuals, (2) *Expert-surprise*, where

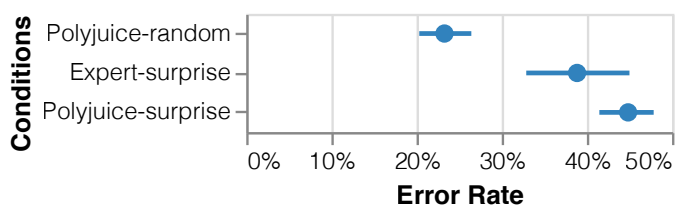


Figure 4.4: Simulation error rates per condition (higher the better). POLYJUICE-*surprise* has the highest error rate, indicating these counterfactuals would add the most information to users if displayed.

⁸Details in Appendix B.3.1.

two graduate students (non-participants) were given access to the model and instructed to create counterfactuals that are surprising given the associated SHAP scores, and (3) POLYJUICE-*surprise*, which uses the selection procedure described in the previous paragraph.

We recruited 13 participants (graduate students with experience in model explanation), and had them analyze the aforementioned *QQP* model. In each round, users were shown an example, the model prediction, and a SHAP explanation, as in Figure 4.3A. Users were instructed to create up to 10 counterfactuals in order to better understand model behavior around the example, for which model predictions were given (users created 6 on average). Finally, users simulated what the model would do on six counterfactuals [Hase and Bansal, 2020], two from each condition (in random order). Counterfactuals where users make mistakes are preferable, as displaying these would add information that users do not already have.

As shown in Figure 4.4, humans simulated model behavior on POLYJUICE-*surprise* counterfactuals only slightly better than random guessing ($45\% \pm 6\%$), *i.e.*, these examples display model behavior that is surprising to users even after seeing explanations and creating their own counterfactuals. *Expert-surprise* also had a high error rate, but at a much higher cost: generating these for just 20 original instances took 1.5–2 hours of expert labor.

While high error rates could be achieved with unrelated or nonsensical examples, all counterfactuals under evaluation were close to the original examples, when measured by syntactic tree edit (≈ 1.0) or Levenshtein distance (≈ 0.2), POLYJUICE-*surprise* being the closest on both. An independent rater labeled 95% of POLYJUICE-*surprise* counterfactuals as “likely written by a native speaker,” in contrast to 85% for *Expert-surprise*, indicating that experts sometimes resorted to ungrammatical or nonsensical sentences to find surprising behaviors.

Qualitatively, the study participants tended to create counterfactuals by perturbing the token with the highest weights (84% of their \hat{x} perturbed tokens in the top 15% quantile of weights), not gaining a real understanding of how the other tokens impact predictions. Participants also made a significant number of mistakes even for tokens they had inspected, *e.g.*, a participant perturbed the example in Figure 4.3A by replacing **help** \rightarrow **play with**, yielding a *Non-Duplicate* model prediction. When faced with **help** \rightarrow **find** in Figure 4.3D, they incorrectly assumed the behavior would be the

same.

These results indicate that POLYJUICE counterfactuals complement feature attribution explanations by displaying information that users often miss, even after they have manually explored the model behavior *beyond* explanations. Moreover, POLYJUICE counterfactuals for this application were more surprising and fluent than *Expert-surprise*, despite being computed automatically.

4.5 Interactive Analysis

While our use of POLYJUICE has so far relied on automatic selection of counterfactuals, we show in this section how an analyst can benefit from *multiple* counterfactuals per x , make use of controlled generation for more advanced analysis, and extract general patterns from individual observations. Our use case is counterfactual error analysis (again, as introduced in Chapter 2) of RoBERTa finetuned on *NLI* (used in §4.3.1), although the techniques are generally applicable.

There is a known correlation between the label *Contradiction* and hypotheses with negation in *NLI* datasets [Gururangan et al., 2018b], which may cause models to fail on non-contradiction negations. We explore this in Figure 4.5A by generating counterfactual hypotheses for a random *Neutral* instance, conditioning only on the original x and the negation control code. While the first two counterfactuals display this failure mode, there is a surprising inconsistency in model behavior between “not” and “n’t”. We note that manual analysis may not explore these three negation forms, and thus not surface this puzzling behavior.

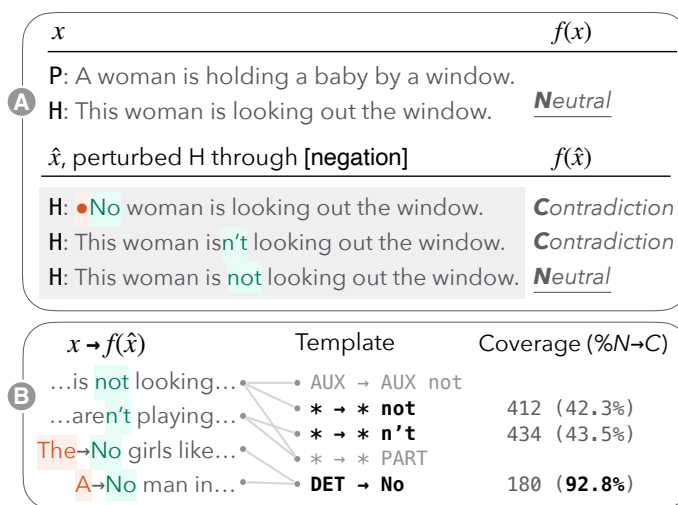


Figure 4.5: (A) An *NLI* case with a *Neutral* prediction (underlined $f(\hat{x})$ are correct). POLYJUICE generates counterfactual hypotheses conditioned on the negation control code. (B) Generalize perturbations into patterns (Chapter 3). **DET** → **no** flips 92.8% of predictions from *Neutral* → *Contradiction*.

To verify if the pattern is widespread, we generate counterfactuals with the negation control code for a random set of instances correctly predicted as *Neutral* ($n = 895$). To generalize individual changes into patterns, we extract frequent *counterfactual templates* with Tempura (Chapter 3, details in Appendix B.4.2),

shown in Figure 4.5B. The top templates (in bold) show that the model flips its prediction from *Neutral* to *Contradiction* with roughly the same frequency ($\approx 43\%$) whether the negation word is “not” or “n’t”, but flips much more frequently with a different negation pattern where a determiner is replaced with “no” (92.8%). While these behaviors may be correct in some instances, they often are not (*e.g.*, Figure 4.5A), and thus would warrant further exploration, and potential mitigation strategies (*e.g.*, counterfactual training, §4.3). Tangentially, the impact of **DET** \rightarrow **no** might lead the analyst to explore the impact of perturbing the *subject* of hypotheses, which we do in Figure 4.6 by placing a [BLANK] on the subject rather than using a control code. This leads to the discovery of unstable and erroneous behaviors regarding *quantifiers*, which we analyze in more detail in Appendix B.4.1.

\hat{x} , perturbed H with [BLANK]	$f(\hat{x})$
[BLANK] looking out the window.	
H: • Two women are looking out the window.	<i>Neutral</i>
H: • Ten women are looking out the window.	<i>Contradiction</i>
H: • More than one person...window.	<i>Entailment</i>

Figure 4.6: Perturbing the subject of x in Figure 4.5A through [BLANK], resulting in erroneous predictions for different *quantifiers* (all should be *Neutral*).

Discussion. POLYJUICE is a powerful tool for interactive analysis. Generating multiple counterfactuals per instance leads to insights that might be missed by manual analysis, and the steering provided by control codes and [BLANK]s allow for analyses that would be non-trivial to do manually [Wu et al., 2019a] or with masked language models (*e.g.*, Figure 4.5B places negations in various parts of sentences, and Figure 4.6 replaces spans with other spans of varying lengths). Besides error analysis, an analogous interactive use of POLYJUICE may be suitable for test creation [Ribeiro et al., 2020] and forms of data augmentation that are more controlled than what we presented in §4.3.

4.6 Related Work

Some prior work in training and evaluation relies on humans to generate counterfactuals from scratch [Gardner et al., 2020, Teney et al., 2020, Kaushik et al., 2020]. Our experiments in §4.3 indicate that asking humans to *label* POLYJUICE counterfactuals yields similar or better results at a lower cost, which motivates an exploration of a mixture of manual and semi-automated generation. Similarly, prior work on analysis relies on experts to create individual counterfactuals or perturbation functions [Wu et al., 2019a, Ribeiro et al., 2020]. In §4.5, we show that POLYJUICE enhances current practice by generating multiple counterfactuals that might have been overlooked, and by providing abstractions that allow for new kinds of analyses.

Prior work on automatically generating counterfactuals typically has a narrower scope in terms of the relationships $x \rightarrow \hat{x}$. For example, adversarial generators aim to maintain semantics while changing model predictions [Ribeiro et al., 2018, Iyyer et al., 2018, Li et al., 2021], whereas concurrent work to our own [Madaan et al., 2021, Ross et al., 2021] automatically generates \hat{x} that change predictions for explanation or analysis, with no constraints on semantics. However, as shown in §4.3–§4.5, a *mix* of label-preserving and label-flipping counterfactuals generated by POLYJUICE is quite useful for training, evaluation, explanation, and analysis. Further, general-purpose counterfactuals may lead to serendipitous discoveries (§4.5), especially as POLYJUICE is not fine-tuned to the target domain (and thus less liable to merely replicate what is already there). Finally, by allowing control through control codes and [BLANK]s, POLYJUICE supports human-generator collaboration, where a person specifies desired changes (*e.g.*, perturb *the sentence subject*). Such collaboration is hard to imagine using automatic generators with no control, or with coarser control through predefined style attributes or labels [Madaan et al., 2020, Malmi et al., 2020]. To our knowledge, prior work on controlled generation [Keskar et al., 2019, Dathathri et al., 2020] does not address *counterfactual* generation.

4.7 Conclusion and Future Work

We propose POLYJUICE, a general-purpose generator that produces fluent and diverse counterfactuals, allowing for control over the kinds and locations of perturbations. With simple, *task-specific* selection heuristics, POLYJUICE supports various downstream tasks on different domains, including counterfactual data augmentation, contrast set generation, counterfactual explanation, and error analysis.

While POLYJUICE is broadly applicable, it is not bias-free: control codes are pre-defined and certainly not exhaustive, and the model is fine-tuned on a collection of paired datasets where certain perturbations are more or less likely (*e.g.*, we observe that words with negative sentiment tend to be slightly more likely than positive ones in some contexts). Collecting naturally occurring counterfactuals is an important area of future research, as is the development of generators that allow for control even without *a-priori* control codes.

Besides improving the generators, further work is needed to improve the value of counterfactuals. For example, while POLYJUICE shows consistent gains across tasks in data augmentation, the improvements on some datasets are not as significant. This aligns with observations in prior work that even manual counterfactuals can be marginally beneficial [Kaushik et al., 2020, Huang et al., 2020b], possibly because the original data is already diverse enough, or the perturbed signal in counterfactuals is too subtle to affect the model (*e.g.*, when only a single word is changed in a long sentence.) We hope to perform more thorough experiments on tuning the amount and the distribution of counterfactual augmentation, as well as other ways of incorporating counterfactuals, such as having explicit terms in the loss function for contrasting counterfactuals with original data [Teney et al., 2020], or other forms of contrastive learning.

Although our applications all involved people, the human-POLYJUICE collaboration in labeling and explanations could benefit from richer interaction mechanisms. We believe POLYJUICE motivates future research on more expressive forms of counterfactual training, where users generate counterfactuals together with POLYJUICE, and label counterfactual *patterns* rather than individual instances. Similarly, interactive explanations and analysis are exciting directions, especially as we

develop new ways of selecting, presenting, and aggregating counterfactuals for various analysis objectives. We explore closer collaborations between generators and humans in the next chapter (Chapter 5), specifically focusing on the potential of using language models to inspire humans considering more diverse patterns as they define their intended modeling tasks.

Chapter 5

AUGMENTING EXPERTS VIA FEW-SHOT EXAMPLE SELECTION FOR EFFECTIVE IN-CONTEXT LEARNING

The emergence of large language models (LLMs) introduces new possibilities for developing customized functions without training models from scratch, but the effectiveness of in-context learning depends significantly on the demonstrative examples in the prompt — the few examples users create alone often contain repetitive patterns or are prone to labeling errors. In response, we design ScatterShot, an interaction mechanism that helps people craft effective in-context learning examples. We use the LLM both as a selector for identifying examples that are worth human feedback, and also as an imperfect annotator that may suggest useful patterns beyond what humans might have imagined. As humans evaluate and fix outputs on these examples and add into the in-context example bucket, ScatterShot helps them iteratively expand the in-context example set with not-yet-learned examples and less error-prone labels. In preliminary experiments, we saw people being able to give more rounds of feedback and getting inspired by LLM outputs, eventually leading towards better coverage of corner cases in their in-context learning tasks.

This work is currently on going, in collaboration with Marco Tulio Ribeiro, Dan Weld, and Jeffrey Heer.

5.1 Introduction

While previous chapters mostly focus on scenarios where developers train or build bespoke, single-purpose AIs, the emergence of large language models (LLMs) introduces new possibilities for developing customized functions (or completing customized tasks) without training models from scratch. Pre-trained on a large amount of text data, models like GPT-3 [Brown et al., 2020] and Jurassic-1 [Lieber et al., 2021] encode enough information to support *in-context learning* [Xie

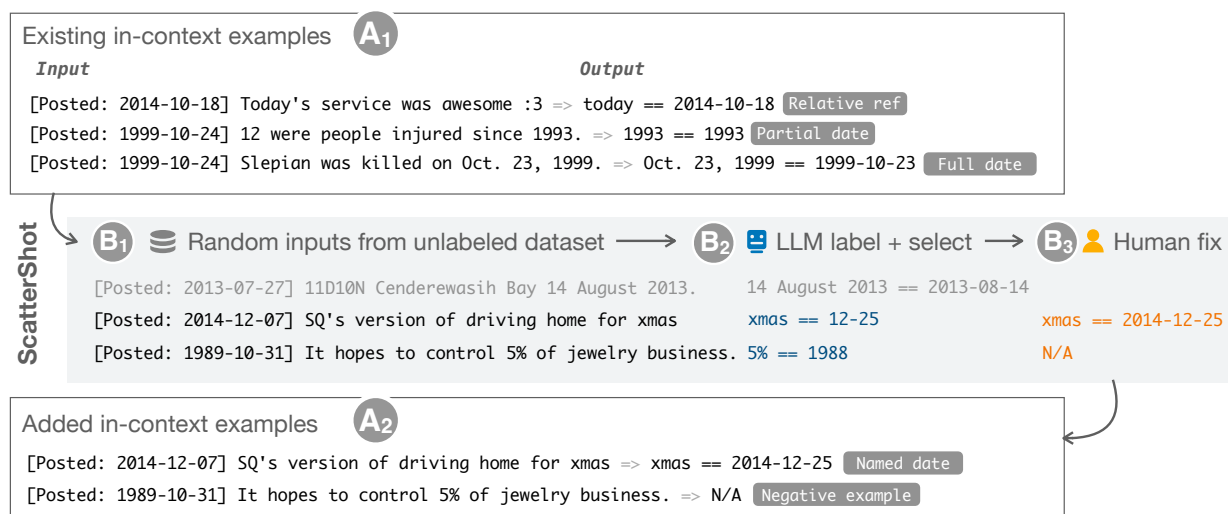


Figure 5.1: An overview of how humans and LLMs can use ScatterShot to iteratively create in-context functions for temporal expression extraction and normalization, with more diverse training sets, and more accurate annotations. The function extracts phrases with temporal meaning from sentences (e.g., “today” in “Today’s service was awesome :3”), and normalizes them into standard formats (“today == 2014-10-18”). Given an existing state of in-context training examples that is likely *under-specifying* the intended functionality (A₁), ScatterShot uses the set to drive an LLM (e.g., GPT-3) to suggest (possibly noisy) annotations to randomly selected unlabeled examples (B₁), filter the ones that are likely to contain not-yet-demonstrated patterns (B₂), such that humans can correct the suggested annotation (B₃), add these correct input-output pairs back to the in-context training set (A₂), and start over.

et al., 2021]: they can be easily customized at run time (without any re-training needed) to handle new tasks, simply by taking in natural language instructions (*prompts*) constructed with a few examples of the task at hand. For example, one could adapt an LLM to act as a translation engine, simply by prepending a few examples of the demonstrative input-output pairs, before the desired input: “English: How are you? French: Comment allez-vous? English: Welcome to Paris! French: Bienvenue à Paris! English: Hello! French:” The LLM will then compute the most likely continuation — a translation for the input “Hello!”, which could be “Bonjour!”. Beyond this toy example, prompting has been used to achieve various ML functionalities in real-time, including

code generation, question answering, creative writing, and others [Swanson et al., 2021, Mishra et al., 2021, Brown et al., 2020].

Although in-context learning has potential, it also presents quality control challenges - the usefulness of a learned function heavily depends on the five to thirty in-context examples that demonstrate the intended behavior [Rubin et al., 2021, Lu et al., 2021]. To improve the in-context learning quality, prior work has looked into automatically searching for the most effective examples for a given use case [Liu et al., 2021b]. However, such methods are only applicable when there is ample labeled data available, which is not usually the case when human annotators wish to define their own functionalities. In the absence of a labeled pool, in-context example authors usually resort to manual crafting of some examples [Jiang et al., 2022], which can lead to labeling errors [Northcutt et al., 2021]. More importantly, as mentioned in Chapter 4, humans can easily get trapped by systematic omissions. Most context-based learning requires less than thirty (and as few as three) demonstrative examples. Since annotators tend to focus on the most obvious patterns they can imagine [Gururangan et al., 2018b], they rarely cover corner cases within their limited budgets, thereby *under-specifying* their intended functions. For example, in Figure 5.1, to build an in-context function that can extract and normalize the temporal information from a sentence [Chang and Manning, 2012], users may cover certain patterns like resolving relative date references (“today” in Figure 5.1A₁) or normalizing common expressions (“Oct. 23, 1999”), but they may easily miss named dates (“xmas” in Figure 5.1A₂). The above issues encourage us to explore training example search in an unlabeled, generative space: how can we help people find examples that can help appropriately specify the in-context function behavior, and how can we guide people in labeling them correctly?

In light of Chapter 4’s observations that generators can support human-generator co-labeling, we create an interactive interface, ScatterShot, to help practitioners craft in-context examples. Underlying ScatterShot is a “collaboration” between a human annotator and a LLM, illustrated in Figure 5.1. It mainly offers two types of assistance: First, to collect in-context examples that more comprehensively define the targeted function, ScatterShot uses an approach akin to active learning to estimate whether a particular input coupled with the correct output will be a valuable

addition to the existing in-context bucket. Second, to make the in-context examples less prone to labeling errors, ScatterShot uses the LLM to generate candidate, potentially noisy labels per input so that users only need to verify and correct the automated label, rather than processing the input from scratch. As the annotator evaluates and fixes outputs on these examples and adds them to the in-context example set, ScatterShot helps them iteratively expand the in-context example set with not-yet-learned examples.

We evaluate ScatterShot in two use scenarios, both concerning perturbing an input text into the output text: (1) a data wrangling task extracting and normalizing temporal expressions from sentences, and (2) a perturbation task for evaluating NLP model consistency. Through preliminary studies, we saw that quantitatively, the in-context example set people create together with ScatterShot achieves better performance than randomly selected examples on held-out datasets. Qualitatively, we saw people being able to give more rounds of feedback and get inspired by current model outputs, eventually leading towards better coverage of corner cases. In other words, we see ScatterShot as a promising way of assisting active and iterative in-context learning.

5.2 ScatterShot: Design and Implementation

5.2.1 Design Considerations

The goal of ScatterShot is to help human annotators find appropriate examples for specifying their desired in-context functionality. An example would be “appropriate” if it helps to:

- D.1 **Specify function applicability.** Because LLMs are generative in nature, they would try to produce the most plausible “label” even for negative examples [Sobieszek and Price, 2022]. For example, in Figure 5.1, LLM attempts to normalize “5%” to a most plausible “1988” even though there is no date in the sentence. This implies that we need to explicitly specify negative examples in the training set, so as to guide LLMs to avoid undesired hallucinations.
- D.2 **Specify intended behaviors on different kinds of input.** It is usually not trivial to anticipate all possible input patterns, include them in the positive training examples, and define desired behavior for each. Therefore, we value demonstrative examples that demonstrate patterns

different from the existing examples.

D.3 Avoid repetitive patterns. There are only a limited number of in-context demonstrations allowed in LLMs due to their computational costs. Therefore, we should avoid wasting budget on repetitive patterns. In ScatterShot’s iterative setting, this also means specifying a stop condition, where the user can dynamically determine whether a function is satisfying without adding any more in-context examples.

To support these specificity needs, we design ScatterShot to use LLMs for two purposes: (1) It ranks and selects diverse unlabeled examples that can help specify function applicability and coverage, and (2) it proposes automatic labels for the selected candidate examples that help people to judge the quality of the current in-context function, and to consider creative labeling patterns they might have missed. Below, we first describe how a human user can interact with ScatterShot, and then explain the underlying workflow, where we select the candidate examples in an active learning style approach.

5.2.2 Interactive Interface

We present ScatterShot as an intuitive interactive interface as shown in Figure 5.2. The interface starts with a task description (Figure 5.2A) and existing in-context learning examples each presented as an input-output pair (B). Both the description and the examples are editable, and are automatically translated to a prompt for driving the LLM. As in Figure 5.2E, the prompt starts with the task description, then involves a series of example pairs in the same format of » [example input] => [example output], then » [candidate input] =>, so the LLM can provide the potential output.

Below existing examples, ScatterShot uses a LLM to propose a batch of candidate examples for users to verify and provide feedback (C). These candidate examples are displayed the same way as training examples. Users have the options to edit the model outputs (*e.g.*, changing from “xmas == 12-25” to “xmas == 2014-12-15”), and adding examples to (or removing them from) the in-context training set (D). Notably, to help specify applicability, ScatterShot allows users to include not only positive examples (*i.e.*, eligible inputs and their desired outputs), but also negative

The screenshot displays the ScatterShot interface. At the top, a task description (A) reads: "Extract all the dates as detailed in the ISO format YEAR-MONTH-DAY." Below this is a section for "EXISTING FEW-SHOT EXAMPLES" (B) with a "PREVIEW THE PROMPT" button and a "COUNT: 4" indicator. This section contains four rows of examples, each with a "D" (description) and "P" (prompt) column, and a control panel with "+", "O", and "-" buttons. The examples are:

- Example 1: D: "[Posted: 2014-10-18] Today 's service was awesome :3"; P: "Today == 2014-10-18"
- Example 2: D: "[Posted: 1999-10-24] 12 were people injured since 1993"; P: "1993 == 1993"
- Example 3: D: "[Posted: 1989-10-31] Slepian was killed on Oct. 23, 1999"; P: "Oct. 23, 1999 == 1999-10-23"
- Example 4: D: "[Posted: 1989-10-31] It hopes to control 5% of jewelry business"; P: "N/A"

 Below the examples is a "CANDIDATES" section (C) with a "LOAD A NEW BATCH" button and a "REMAINING INSPECTION BUDGET: 100" indicator. It shows one candidate row:

- Candidate 1: D: "[Posted: 2014-12-07] SQ's version of driving home for xmas"; P: "xmas == 12-15"

 To the right of the interface, a raw text prompt (E) is shown, which is a concatenation of the task description and the prompts from the existing examples in the order they appear:


```
Extract all dates as detailed in the ISO format YEAR-MONTH-DAY.
>> [Posted: 1999-10-24] 12 were people injured since 1993. => 1993 == 1993
>> [Posted: 1989-10-31] Slepian was killed on Oct. 23, 1999 => Oct. 23, 1999 == 1999-10-23
>> [Posted: 2014-10-18] Today's service was awesome :3 => today == 2014-10-18
>> [Posted: 1989-10-31] It hopes to control 5% of jewelry business. => N/A
>> [[CANDIDATE INPUT]] =>
```

Figure 5.2: The ScatterShot interface, with (A) task description, (B) existing in-context examples, and (C) candidate examples awaiting human inspection. (D) Users can make edits to LLM outputs, sort the candidates into positive demonstrative examples (+), negative ones (-), or just not include the candidate (O). The description and the examples get transformed to raw text prompts. One set of in-context example produces multiple prompts depending on how the examples are ordered; (E) shows a prompt with one possible ordering.

examples (*i.e.*, ineligible inputs and a placeholder output “N/A”). Besides taking human feedback, we also use the candidate set to implicitly support users self-assessing the quality of the current in-context dataset/function: If all unedited LLM outputs seem reasonable on some consecutive batches of selected candidates, it is likely that the in-context examples have achieved reasonable coverage, and that users can decide to terminate the iteration process.

The interface is mostly task-agnostic such that it can be generalized to various use scenarios that involve one-on-one mapping between inputs and outputs. It can be easily invoked in Jupyter Notebook and therefore can seamlessly support users’ natural workflow.

5.2.3 Data Slice based Sampling

Underlying the interface, the key idea of ScatterShot is to select candidates that are non-repetitive, diverse, and worth human feedback. In essence, it is an active learning problem [Settles, 2009,

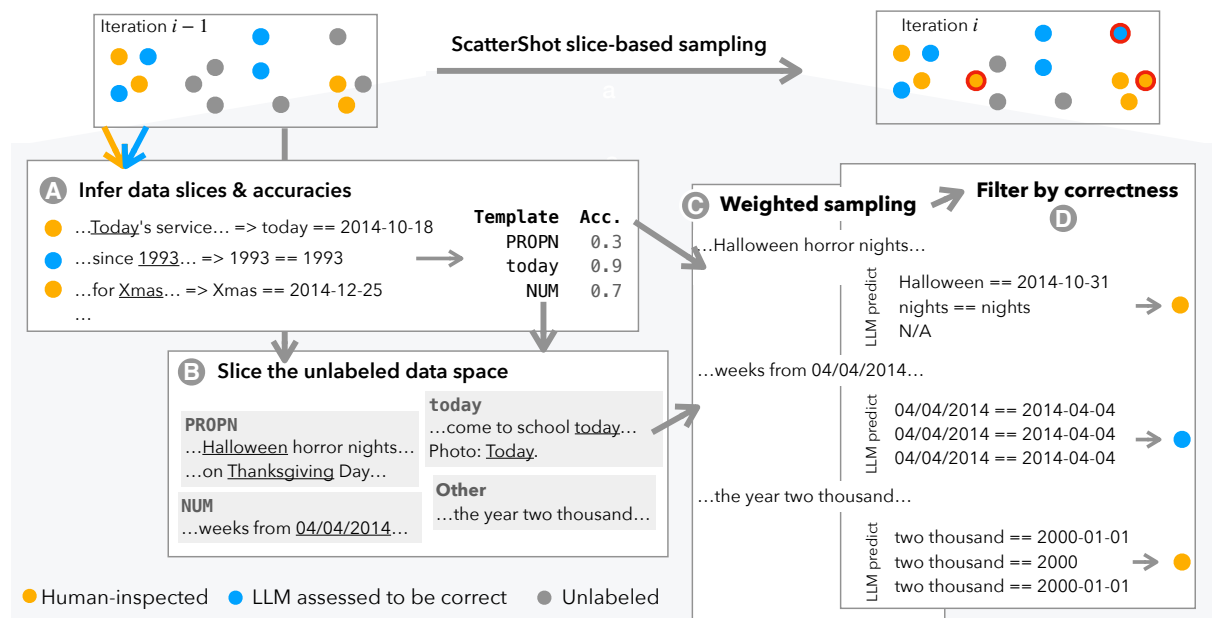


Figure 5.3: A high level overview of ScatterShot’s slice-based sampling. We use the data status in the $i - 1$ -th iteration to perform sampling for the i -th iteration: (A) we use the already LLM- or human- assessed examples to infer data slices represented as templates and estimate how the current in-context function performs on different slices;(B) we use these templates to partition the unlabeled data space into various groups (phrases that match the templates are underlined); (C) we sample unlabeled examples based on their corresponding slice size and estimated accuracy, to prioritize sampling from more uncertain and larger slices; (D) we run the current in-context function on the selected inputs, and deem the function correct, if prompts with different in-context example orders produce the same output. An example will only be surfaced to the interface for human inspection if the function is incorrect on it (annotated with the orange dot).

Tran et al., 2019] where a model dynamically queries an annotator for the desired output for these selected examples, so that the model can improve its performance using the updated training data. One typical strategy is uncertainty sampling, *i.e.*, to prioritize examples that the model is least certain how to label according to model confidence or the entropy of all possible labelings [Lewis and Gale, 1994]. Due to the generative nature of LLMs, there are an indefinite number of possible outputs (or labels), making it harder to estimate model confidence. Even if we can measure the confidence, it is still too computationally expensive to run the LLM on every candidate input.

As a workaround, we try to estimate how an in-context function will respond to *a group of*

similar examples instead of individual ones, through slice-based [Chen et al., 2019] sampling. By dividing the data space into subsets based on inter-instance similarities, we estimate how well the current in-context function is performing per slice, and we prioritize selecting and labeling data points from slices that haven’t yet been learned. More formally, as we iteratively select our in-context examples, at the i -th iteration, we split the current unlabeled data space $\mathbf{X}_i^u = \{x_1, \dots, x_n\}$ into a series of m automatically inferred data slices $\mathbf{S}_i = \{S_{i,1}, \dots, S_{i,m}\}$. We estimate the data slice accuracy on slice j , $A_{i,j} \in [0, 1]$, using the samples processed in the previous two iterations. Given an unlabeled input x that belongs to a certain slice $x \in S_{i,j}$, we assign its probability of being sampled proportional to the estimated performance $A_{i,j}$ (such that we prioritize uncertain examples), multiplied by the size of the slice $|S_{i,j}|$ (to encourage coverage of the unlabeled space X_i^u): $P(x) \propto |S_{i,j}| \cdot (1 - A_{i,j})$. We further filter the select-to-label examples, \mathbf{L}_i , by estimating in-context function correctness on them, and present examples that are worth human feedback, as well as the LLM-proposed labels, in the interface.

The slice-based approach can be viewed as uncertainty sampling of different input types. It addresses the aforementioned design rationales, *i.e.*, interactively identifying data slices helps specify the function applicability, and prioritizing unlearned data slices alleviate the risk of wasting the labeling budget on repetitive patterns. Below, we introduce how to infer the slices \mathbf{S}_i , estimate slice performances, and filter selected examples by correctness.

Infer data slices. Perhaps the most straightforward way would be to automatically cluster the unlabeled input examples based on their similarities. However, empirically, we see that vanilla similarity measurements cannot help reveal meaningful slices. For example, if we run clustering based on cosine similarities in the input sentence embedding space [Reimers and Gurevych, 2019], then “Photo: Today.” is considered closer to “Photo: on Flickr.” (similarity score $s = 0.48$) than to “Are you going to yoga today?” (similarity score $s = 0.29$), though the latter shares the key temporal phrase of “today”. As an alternative, we propose to estimate slices more explicitly using heuristically created templates, similar to the Snorkel labeling functions [Ratner et al., 2016]. We collect all the inputs that have been sampled (and thus processed by the LLM) up until iteration $i - 1$, extract trigrams from the sentences, and select representative templates using Tempura (Chapter 3) –

so both “Photo: Today.” and “Are you going to yoga today?” can be put in a cluster of “today”.

We then sort the unlabeled data points into slices. If an unlabeled input $x \in \mathbf{X}_i^u$ contains a phrase that matches with a template $t_{i,j}$, then we say x belongs to the corresponding slice $S_{i,j}$. If x is not matched with any of the m templates, then it belongs to a single, unsorted slice of $S_{i,m+1}$.

Note that to further eliminate potential noise in extracting trigram templates, we take advantage of the nature of our perturbation tasks, and only use trigrams in the *key phrases*. Given a labeled, positive example, *i.e.*, a pair of original and perturbed sentences (x, y) , we deem the edited phrases of x to be key to the task, if most x remain unchanged (Levenshtein edit distance $d \leq 0.5$), and the remaining phrases to be key otherwise. For negative examples where the output would be a plain “N/A”, we simply use the full sentence as the key phrase.

Estimate data slice performance. The first step to quantifying the performance of an in-context function on a data slice is to determine when a function is considered correct on a single, unlabeled input. To do so, we take inspiration from Quey-by-Committee [Melville and Mooney, 2004] and the unanimity principle [Khani et al., 2016]. Specifically, following Lu et al. [2021]’s observation that the order of the same in-context examples can drastically change LLM performances, we form three different prompts by randomly reordering the examples, and get three outputs accordingly. We only consider the current in-context function *correct-by-unanimity* on an input x , when multiple different prompts formed with the *same* in-context examples predict the same output.

To track the most recent progress, we estimate model performance using the examples processed in the last two iterations. We deem an example $x \in \mathbf{L}_{i-1} \cap \mathbf{L}_{i-2}$ correct, if it is correct-by-unanimity, or the human annotator did not interact with x ’s candidate output in the interface, indicating x has passed the manual inspection. We update the performance of the corresponding slices $\{s_{i,j} | x \in s_{i,j}, x \in \mathbf{L}_{i-1} \cap \mathbf{L}_{i-2}\}$ based on these correctness.

Filter selected examples by correctness. To maximize the utility of human inspection, we further filter the current samples L_i using correct-by-unanimity, and only present the examples to humans if the outputs from the three prompts disagree. We choose the output with the minimum perplexity score as the LLM-provided, candidate label.

5.3 Case Studies

We quantify the effectiveness of ScatterShot via preliminary case studies on two tasks. We first use a temporal expression normalization task to estimate whether the ScatterShot-learned functions are high quality enough. Then, we also describe qualitatively how ScatterShot aids humans to articulate their desired functions, through another task of perturbation function creation.

5.3.1 Case 1: Temporal Expression Extraction and Normalization

The task relates to data wrangling [Verbruggen et al., 2021], and concerns extracting phrases with temporal meaning from sentences or documents, and normalizing them into standard formats [Chang and Manning, 2012]. As shown in Figure 5.1, these temporal expression can include absolute dates or dates relative to a reference (*e.g.*, post date), and can have different granularity (*e.g.*, exact date vs. year only). We take the data from [Almasian et al., 2021], which contains common temporal datasets including TimeBank [Pustejovsky et al., 2006] and TweepTime [Tabassum et al., 2016], containing temporal expression annotations on news articles and tweets, respectively. This fully labeled dataset enables us to simulate a labeling process with an oracle that always provide groundtruths. Moreover, by evaluating the in-context function on holdout sets, we are also able to quantify the learned function quality.

Since some of the normalizations are arguably ambiguous and hard to control, we process the dataset such that (1) we only kept data points that could be normalized to the format of YYYY-MM-DD, and (2) we split long documents into single sentences, and only kept sentences that involve absolute dates, or describe dates relative to the document publication date, or no time expression at all (as the pool for negative examples). We collected 491 positive examples and 369 negative ones this way.¹ We randomly sample 100 positive and 50 negative examples to form the test set, and use the remaining examples as a pool for selecting in-context examples.

Procedure and baseline. We take advantage of the gold labels, and design a simulation study comparing *ScatterShot* with a *Random* baseline where we replace the slice-based sampling with

¹The processed data can be found in <https://github.com/tongshuangwu/scattershot-data>.

Conditions	Extraction			Normalization		
	F1	Precision	Recall	F1	Precision	Recall
Random	58.6	57.5	61	48.9	48	51
ScatterShot	69.8	69	71	58.6	58	60

Table 5.1: Quantitative results comparing ScatterShot with random baseline on temporal expression. ScatterShot outperformed the baseline on all metrics.

plain random sampling.

The simulation setup for both systems is the same, and the gold labels are used as the oracle labeler. The process starts with three random seeding examples, and in each iteration, we compare the gold label with the LLM-proposed candidate label. If these two labels differ, we add the pair of (input, oracle output) into the in-context example set, simulating the case where the human annotator corrects the LLM; Otherwise, the oracle will not perform any action, simulating cases where humans decide to do nothing on already-learned patterns. We repeat the process until one of the three stop conditions is met: (1) the in-context example set contains more than 30 data points, (2) the oracle labeler has been presented more than 100 examples, or (3) in two consecutive iterations, the in-context function correctly generated outputs for all the selected examples. Once the iteration is stopped, we use the final in-context function to generate outputs for all the examples in the test set, and follow Chang and Manning [2012]’s practice in evaluating the result. Specifically, we report the F1, recall, and accuracy for both the time expression extraction and the normalization. We repeat the procedure three times for more stable observation.

Result. As shown in Table 5.1, ScatterShot outperforms the baseline especially on extraction. Note that this is a stronger baseline than asking humans to manually craft a training set, as it assumes a reasonable access to less obvious cases that humans might have missed, and it also ensures that all the labels are correct. This suggests that ScatterShot has the potential to improve in-context learning function qualities.

Coverage	Input	Q: Are there more girls or boys? A: equal
	Template	(Not applicable)
	Random	Q: Are the girls and boys equal in number? A: yes
	ScatterShot	Q: Are the girls and boys equal? A: yes
Fluency	Input	Q: What does the woman have on her face? A: headband
	Template	Q: Does the woman have on her face headband ? A: yes
	Random	Q: Does the woman have a headband on her face ? A: yes
	ScatterShot	Q: Does the woman have a headband on her face ? A: yes
Correctness	Input	Q: What make is the phone? A: vtech
	Template	Q: Make is the phone vtech ? A: yes
	Random	Q: Is the phone a vtech ? A: yes
	ScatterShot	Q: Is the phone a vtech ? A: yes
Correctness	Input	Q: What monument are they next to? A: unknown
	Template	Q: Are they next to unknown? A: yes
	Random	Q: Is the monument unknown? A: yes
	ScatterShot	Q: Are they next to an unknown monument? A: yes
Correctness	Input	Q: What type of motorcycle is in the picture? A: mountain
	Template	Q: Is the mountain in the picture ? A: yes
	Random	Q: Is the mountain type of motorcycle in the picture ? A: yes
	ScatterShot	Q: Is the motorcycle in the picture a mountain bike? A: yes

Table 5.2: Example outputs from perturbation functions built in ScatterShot and Random condition, and from templates in Ribeiro et al. [2019] et al. ScatterShot functions tend to be able to perturb more forms of inputs (“coverage”), sound more natural (“fluent”), and tend to produce desired outputs (*i.e.*, the new question-pair are logically equivalent to the original pair).

5.3.2 Case 2: Targeted Text Perturbation

We also explore using ScatterShot to replicate perturbation functions in prior work for evaluating NLP model behavior consistency. Given one testing objective, researchers usually investigate a lot of manual effort to create and refine multiple templates for handling different input types. Even with these, the templates could be too strict to generate natural texts for many intended inputs. For example, just for testing question answering models’ robustness on logically equivalent questions, Ribeiro et al. [2019] et al. created four subtly different templates that perturb based on their parsing tree structures. We are interested in whether ScatterShot can help achieve compatible or better

perturbations while saving the manual effort crafting templates. As such, we design the second case study to replicate the *logical equivalence* perturbation in [Ribeiro et al., 2019] with in-context learning.

Procedure. While we were able to download Ribeiro et al. [2019]’s pairs of input sentences and template-perturbed outputs², the template outputs are generally noisy and therefore cannot be an oracle as in the temporal expression case. Rather, we use the case to run a preliminary user study. We invited two NLP graduate students to interact with the ScatterShot interface and build the perturbation function for *logical equivalence*. We first briefed participants on the task by providing them with task descriptions and sample (seeding) examples. Then, we asked them to create in-context training sets twice, once with each condition (ScatterShot and Random, where again we provide ScatterShot to users without the slice-based sampling), whose orders were counterbalanced. We asked them to spend up to 25 minutes per condition, but allow them to stop early if they are satisfied with their functions. We encouraged participants to think aloud and describe their actions as they completed the task.

Results. We took their built in-context function and ran a similar quantitative comparison as in the previous temporal expression study. We saw that the function they built through ScatterShot produces more valid perturbations (*i.e.*, correct and fluent ones for 78% of the time) than the function produced in the *Random* condition (61%).³

Qualitatively, *Random* functions contained 21 in-context examples on average whereas *ScatterShot* ones contained 30. This indicates that without the help of ScatterShot, users tended to misjudge the in-context function quality, and incorrectly stop the iteration way before the function performance converges. As shown in Table 5.2, the use of ScatterShot also helped increase coverage on less apparent data slices like “how many” and “why” questions, rather than the most frequent “what” questions.

We also compared *ScatterShot* functions with Ribeiro et al. [2019]’s original templates, and

²https://github.com/marcotcr/qa_consistency

³We resolved to manually verifying perturbations on 120 original examples because, again, the template outputs were noisy.

found ScatterShot outperformed *Templates* on coverage, fluency, and correctness. While preliminary, these results point to ScatterShot potential on saving human efforts in creating fine-grained functions, alleviating the need of handcrafting templates.

5.4 User Study Planning

Based on the preliminary case studies, we plan to conduct user studies on both the data wrangling and text perturbation tasks. We hypothesize that:

- H.1 Compared to manually defining in-context learning examples, editing from LLMs helps cover more diverse input patterns.
- H.2 Inconsistency sampling helps (1) people better estimate function performance, (2) improve the performance of the built function.

Conditions. We will compare **ScatterShot** with one of the two baselines: (1) **unaided**, where users manually craft prompt without any help from ScatterShot. This simulates the status-quo of people creating their own in-context learning examples in the wild. (2) **random ScatterShot**, where we provide ScatterShot to users without the inconsistency sampling strategy. In this condition, users still receive annotation inspiration from LLMs, but not suggestions on feedback-worthy inputs.

Participants. We plan to recruit 10-15 participants. For the text perturbation task, we plan to recruit NLP researchers who understand perturbation-based model evaluation, whereas data wrangling could be suitable for general audience.

Procedure. The study will use a within-subjects design, and we estimate it will take about up to 90 minutes for each participant. In the time frame, they will perform a randomly selected task (data wrangling or text perturbation). The task session will be similar to the text perturbation pilot study (Chapter 5.3.2), consisting of task briefing and function construction in three conditions.

Data collection. We will observe and analyze three sets of data. First, to compare the functions built in different conditions, we will save their final in-context learning, and apply them to heldout datasets. Second, to assess participants' **self-perceived experience**, we will use a standard seven-point Likert Scale [Likert, 1932] to collect their estimations on the function quality ("how accurate and generalizable do you think is the in-context function?") and their interaction experience (*e.g.*,

“how much are you inspired by the ScatterShot suggestions?”). Finally, to quantify their **interaction mechanisms and behaviors**, we will log their interactions with the system in both conditions. We are particularly interested in how participants react and iterate on model outputs, and we plan to reflect this by inspecting their edits on LLM annotations, their manipulation on the in-context examples, and the number of iterations they are willing to go through. For example, a larger number of iterations in ScatterShot might show that our design find more difficult cases that deserve inspection, and therefore helping humans to more holistically assess their in-context function quality.

5.5 Related Work

Selecting demonstrative examples. In-context learning has been shown to heavily depend on the demonstrative, “training” examples, and LLMs can only perform in-context learning if it has seen the corresponding distribution or concept [Min et al., 2022, Xie et al., 2021, Rubin et al., 2021]. Prior work has explored selecting effective demonstrations, and has shown that because pre-trained models possess high-level semantic features, sampling or active learning tend to help identify informative examples [Tamkin et al., 2022]. Many of these studies tries to optimize for *local* performance, *i.e.*, dynamically selecting the most similar demonstrations for each given input [Liu et al., 2021b], which requires searching and selecting in-context examples from fully labeled datasets [Rubin et al., 2021, Wang et al., 2021b, Zhao et al., 2021]. In contrast, our work focuses more on an unlabeled space, and as a result focuses more on *iterative task definition*.

In the unlabeled search space, Chang et al. [2021] has tried to allocate an annotation budget based on training data diversity akin to sampling approaches in active learning. Indeed, the selection part of our work can be seen as active learning [Settles, 2009, Tran et al., 2019] for in-context learning. The key idea behind active learning is that machine learning models can achieve higher performance with fewer training examples, if it is allowed to choose its own, most informative training examples. Given a budget, an active learner iteratively selects examples-to-annotate from an unlabeled pool according to some ranking mechanism. In Chang et al. [2021]’s case, the examples are selected through diversity [Sener and Savarese, 2017]; However, diversity is usually ill-defined

for various tasks [Rubin et al., 2021]. Our method is closer to uncertainty sampling [Lewis and Gale, 1994], where an active learner queries the instances about which it is least certain how to label. Note that because LLMs are generative in nature and do not have clear probabilistic distributions across all “labels” as in *e.g.*, classification tasks, we estimate uncertainty using the LLM output stability: if different prompts “vote” for different outputs, then we say the LLM is uncertain about the corresponding input given the current few-shot examples. This voting strategy is also quite relevant to Query-By-Committee [Seung et al., 1992] where a list of “committee” models trained on the same labeled set vote on the labelings of query candidates. Importantly, while many empirical results suggest that active learning is effective, it does suffer from certain limitations. For example, the labeled examples are not drawn i.i.d from the underlying data distribution [Settles, 2009], and therefore can sometimes be imbalanced [Pop and Fulop, 2018] or less effective than random sampling [Imberg et al., 2020]. Our method will likely share the same limitations, though we leave it to future work to articulate scenarios where ScatterShot is most useful.

Example creation and selection is also commonly seen in programming-by-demonstration (PBD). For example, Zhang et al. [2020] explored effectively selecting examples that can help disambiguate and validate synthesized regular expressions. We share similar motivations that interactively and iteratively suggesting corner cases help synthesize the right function, but unlike PBD where new examples are always *pruning* the function search space, ScatterShot focuses on *expanding* the function coverage. Therefore, it is essential to select examples that incentivize people to provide feedback.

Generators for inspirations. Prior work has shown that generators can potentially help humans complete tasks more effectively [Wang et al., 2021b, Wu et al., 2021], and compensate for their omissions [Ross et al., 2022, Lee et al., 2022]. Generators seem specifically effective for ideation: they can introduce new characters into human-generator co-written novels [Lee et al., 2022], and can exploit specific linguistic patterns and enrich otherwise rare data types [Liu et al., 2022]. These works mostly use generators for ideating *instances*, with the assumption that the task itself (*e.g.*, novel creation, dataset expansion) is clearly defined. Our work more explicitly explore the gains from *concept* ideation, or how much LLMs can help people refine their task specification.

5.6 Discussion

We design a human-LLM collaboration mechanism to help humans craft diverse and clean in-context learning examples. In preliminary experiments we found that participants were able to provide more feedback and get inspired by LLM outputs, eventually resulting in better coverage of corner cases in their in-context learning tasks. We believe ScatterShot shows the potential of using LLMs to help people refine their task definition, and thereby leading to improved in-context learning generalizability by suggesting and labeling corner cases.

Still, this assumes people only use LLMs when they want to tailor the model for high-quality and reproducible functionalities. In reality, with LLMs flexibly responding to natural language instructions, even *end users* may use these models for their possibly one-off own purposes. In the next chapter, we explore the scenario where *end users* lead LLMs towards their own desired outcomes through interactions.

Chapter 6

END USERS DEBUGGING LLMS: TRANSPARENT AND CONTROLLABLE HUMAN-LLM INTERACTION THROUGH AI CHAINING

Although large language models (LLMs) have demonstrated impressive potential on simple tasks, their breadth of scope, lack of transparency, and insufficient controllability can make them less effective when assisting humans on more complex tasks. In response, we introduce the concept of *Chaining* LLM steps together, where the output of one step becomes the input for the next, thus aggregating the gains per step. We first define a set of *LLM primitive operations* useful for Chain construction, then present an interactive system where users can modify these Chains, along with their intermediate results, in a modular way. In a 20-person user study, we found that Chaining not only improved the quality of task outcomes, but also significantly enhanced system transparency, controllability, and sense of collaboration. Additionally, we saw that users developed new ways of interacting with LLMs through Chains: they leveraged sub-tasks to calibrate model expectations, compared and contrasted alternative strategies by observing parallel downstream effects, and debugged unexpected model outputs by “unit-testing” sub-components of a Chain. In two case studies, we further explore how LLM Chains may be used in future applications.

This work was done in collaboration with Carrie J. Cai and Michael Terry, and was originally published at CHI 2022 paper [Wu et al., 2022b].

6.1 Introduction

While previous chapters mostly focus on aiding quality improvement of NLP systems before deployment, *end users* are the ultimate customers of AI models. Unlike experts, these users usually have more concrete objectives in using a model, but little understanding or access to the underlying

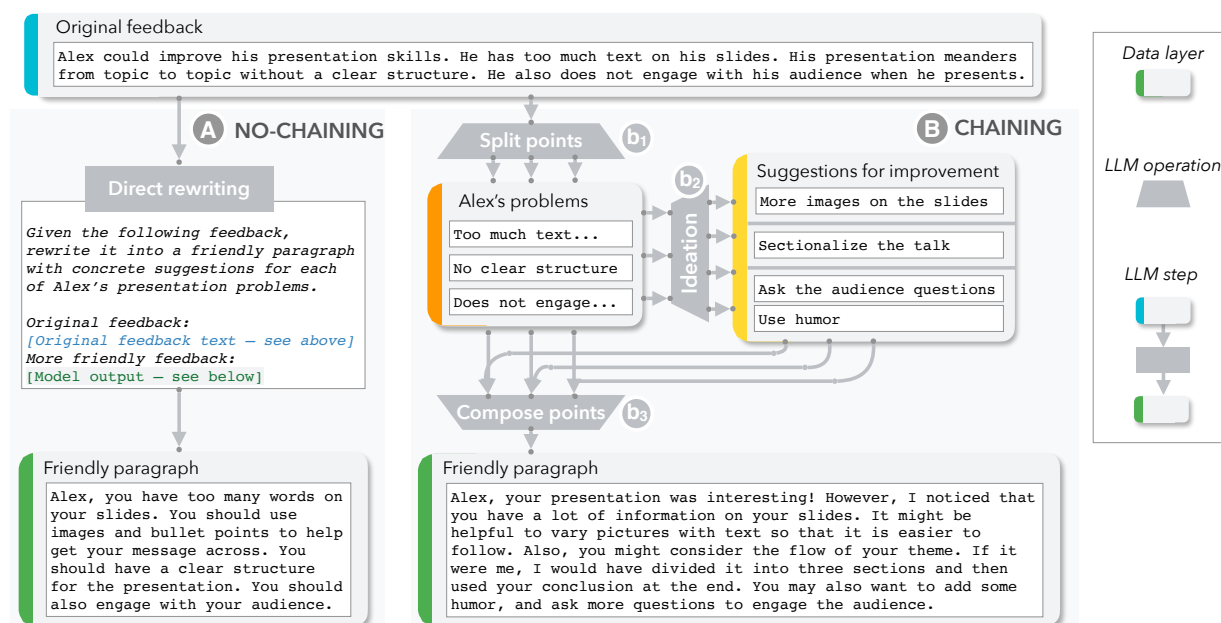


Figure 6.1: A walkthrough example illustrating the differences between *no-Chaining* (A) and *Chaining* (B), using the example task of writing a peer review to be more constructive. With a single call to the model in (A), even though the prompt (italicized) clearly describes the task, the *generated paragraph* remains mostly impersonal and does not provide concrete suggestions for all 3 of Alex’s presentation problems. In (B), we instead use an LLM Chain with three steps, each for a distinct sub-task: (b1) A **Split points** step that extracts each individual presentation *problem* from the *original feedback*, (b2) An **Ideation** step that brainstorms *suggestions* per problem, and (b3) A **Compose points** step that synthesizes all the problems and suggestions into a final *friendly paragraph*. The result is noticeably improved.

model structure, and would need specific supports on interpreting and controlling models even without the domain knowledge. In this chapter, we design interaction technologies to help users recover from errors they see as they collaborate with AIs.

Specifically, we focus on the end user interaction with Large Language Models (LLMs), as the relative ease of natural-language-based prompt programming suggests that LLMs may be useful assistants for real-world tasks, with users customizing the models to their own needs. In this light, recent work in Natural Language Processing (NLP) has begun to examine the algorithmic capabilities of LLMs, mostly on synthesized tasks [Wang et al., 2021a, Floridi and Chiriatti, 2020,

Reynolds and McDonell, 2021]. However, many real-world tasks can be quite complex (*e.g.*, outlining long essays, debugging software code), and may present challenges for current LLMs to solve from a single model run. For example, as LLMs learn the forms of language [Bender and Koller, 2020], they produce lower quality outputs when solving tasks that require multi-step reasoning [Tan et al., 2021, Branwen, 2020, Wei et al., 2022]. Likewise, they may fail to capture the subtleties of many tasks that involve multiple objectives simultaneously (*e.g.*, identifying *and* fixing multiple bugs in a code snippet). Figure 6.1 shows a task involving multiple concurrent objectives: (1) to rewrite peer feedback to be more friendly, and (2) to rewrite it with additional concrete suggestions, and (3) to ensure that each noted sub-problem (*e.g.*, too many words on slides, presentation meaders, does not engage with audience) is addressed. While an LLM can both generate suggestions [Webb, 2021] and adjust the tone in isolation (*e.g.*, in [sty]), it lacks the capability to perform both tasks together well in an end-to-end manner. As a result, it produces a mediocre paragraph that only meets a few requirements (see output of Figure 6.1A).

Besides being inherently limited for complex problems, LLMs are also difficult to interact and collaborate with, as they can be opaque and hard to debug. Since LLMs can take in any natural language prompts, end users may struggle to determine how to change their prompts to remedy unexpected model outputs. They may also have difficulties developing accurate mental models of an LLM’s capabilities and limitations. There are no obvious edits on the prompt that can, for instance, encourage the model to add more suggestions regarding “too much text on slides” in Figure 6.1A.

In this work, we introduce the notion of *Chaining* multiple LLM prompts together, to help users accomplish complex tasks with LLMs in a way that is more transparent and debuggable. Unlike ScatterShot (Chapter 5) where LLMs leads to improve in-context learning generalizability by suggesting and labeling corner cases, we at the scenario where *end users* lead LLMs towards their own desired outcomes through interactions. Chaining takes advantage of LLMs’ unique ability to handle a variety of independent tasks. In a Chain, a problem is broken down into a number of smaller sub-tasks, each mapped to a distinct step with a corresponding natural language prompt; results of one or more previous steps are aggregated in the next step’s input prompt. Thus, Chaining enables users to run *the same* model on multiple sub-tasks, thereby granting each sub-task a higher

likelihood of success (as opposed to solving the entire task in one go). In Figure 6.1B, while the underlying LLM remains the same, by *splitting* (i.e., extracting) presentation problems (b_1) and *ideating* suggestions per problem (b_2), the final *composed* paragraph (b_3) is more comprehensive in addressing all problems, and has a more constructive tone.

In addition to potentially improving outcomes, Chaining opens up new channels for fine-grained human feedback and control. For example, thanks to the separate *Ideation* step in Figure 6.1b₂, Chaining allows users to customize which suggestions to include in the final paragraph, an operation that is unavailable without Chaining (Figure 6.1A). We develop an interactive interface to expose these additional “knobs” to end users. The interface visualizes the Chain structure, and allows users to customize a Chain at various levels: they can iterate on the local prompts in each step, edit intermediate data between steps, or modify the entire Chain. To inform the design of this tool, we surveyed 73 existing LLM use cases and summarized them into a set of *LLM primitive operations*, each with default prompting and data structures. They help inform what types of sub-tasks could be used within a Chain, as well as how those steps can feed into each other.

To evaluate the impact of Chaining on both task performance and user experience, we conducted a within-subject user study, in which 20 participants completed tasks using both Chaining and a standard (non-Chaining) interface, with the same underlying LLM powering all the steps in the Chaining interface, as well as the non-Chaining one. Our results show that Chaining significantly improved key dimensions of the human-AI experience: transparency, controllability, collaboration, and mental support. In addition, participants also achieved higher-quality outcomes ~82% of the time using Chaining. We also saw participants leveraging Chaining for purposes beyond immediate task accomplishment — they calibrated their expectations of the model using the smaller scope of sub-tasks, explored alternative prompting strategies by comparing parallel downstream effects, and debugged unexpected model output by isolating and “unit-testing” different parts of a Chain. Critically, these improvements were achieved *without changing the model itself*. These findings suggest that one way to improve the explainability and debuggability of an otherwise opaque, black-box LLM is to have it *do less*: breaking a problem up into smaller problems, having the model solve each (smaller) problem separately, showing the intermediate results, and allowing users to

edit those results.

The ability to chain LLM calls using a set of Chaining building blocks, within an interactive interface, collectively represents a novel method and system for prototyping new AI-powered tasks and features using LLMs. We conclude the chapter with case studies illustrating how Chaining can support more diverse applications in the future, as well as insights into challenges and opportunities that arose from our experiments. In summary, we contribute:

- We introduce the notion of *LLM Chaining*. Through a series of chained model calls, each targeting a small and well-scoped sub-task, we adapt *a single LLM* to contribute to *multiple sub-components* of a task.
- We design and implement building blocks for constructing and interacting with LLM Chains. These include a set of primitive LLM operations representing functions well-scoped for a single model run, and an interactive interface that displays the intra- and inter-step structures of a Chain. Users can run Chains step-by-step, and customize them at various granularities (editing intermediate model outputs, rewiring steps, etc.).
- We report results from a 20-person evaluation that shows Chaining can increase system transparency, controllability, and task outcomes. Importantly, these gains are achieved without any changes to the underlying model. Combined with the case studies, we demonstrate the potential of improving explainability and debuggability of LLMs through task decomposition and finer-grained application of LLM models.

Taken together, our findings inform the design and research of future human-LLM collaborative systems, an area of critical importance in years to come.

6.2 Background and Related Work

6.2.1 Large Language Models

A generative language model is primarily designed to continue its input with plausible output (*e.g.*, given a prompt “I went to the”, it might auto-complete with “coffee shop”). However, when pre-trained on billions of samples from the Internet, recent transformer-based LLMs [Vaswani

et al., 2017] like GPT-3 [Brown et al., 2020] and Jurassic-1 [Lieber et al., 2021] encode enough information to support additional *in-context learning*: they can be easily customized at run time (without any re-training needed) to handle new tasks beyond text continuation. To invoke the desired functionalities, users need to write natural language instructions, or *prompts* [Lu et al., 2021, Betz et al., 2021, Liu et al., 2021b], that are appropriate for the task. The most common patterns for prompting are either zero-shot or few-shot prompts. Zero-shot prompts directly describe what ought to happen in a task. For example, we can enact English-to-French translation with a prompt such as “Translate the sentence “Do you like the weather?” to French:”. In contrast, few-shot prompts show the LLM what pattern to follow by feeding it examples of desired inputs and outputs: “[English] Hello! [French] Bonjour! [English] Do you like the weather? [French]”. Given either of these prompts, the LLM may respond with the French translation “Vous aimez le temps?” [Jozefowicz et al., 2016]. Importantly, such task customization happens on the fly and, as a result, a single LLM can be flexibly adapted to a wide variety of use cases like code generation, question answering, creative writing, etc. [Brown et al., 2020, Swanson et al., 2021]. This flexible adaptation, together with the text-in, text-out structure, creates an intuitive natural language interface between humans and the model.

Despite their versatility, LLMs require careful prompt design. Various studies therefore focus on *prompt engineering* [Lu et al., 2021, Betz et al., 2021, Liu et al., 2021b]. As manual prompting can be sub-optimal, some work automatically mines more effective prompts. However, the mined prompts tend to be less human-readable [Shin et al., 2020] and therefore less compatible with human-AI interaction. Conversely, strategies like progressive generation (*i.e.*, multi-round text expansion) [Tan et al., 2021] and meta-prompting (*i.e.*, asking the model to elaborate on the problem) [Betz et al., 2021, Reynolds and McDonell, 2021] attempt to seed LLMs to generate more effective prompts before solving the task. In essence, these approaches also adopt the spirit of multi-step problem solving, but focus on expanding the context *without* human intervention. Our work defines Chaining more comprehensively, with primitive operations that illustrate LLM capabilities, LLM steps that can add or remove information along the Chain, and editable intermediate data points.

6.2.2 Human-AI Collaboration

Human-AI interaction has been explored in domains such as classification [Bansal et al., 2021, Smith-Renner et al., 2020], drawing [Oh et al., 2018, Davis et al., 2016], translation [Green et al., 2014], creative writing [Clark et al., 2018, Gero and Chilton, 2019], and design ideation [Koch et al., 2019]. Prior work has noted core challenges of the interaction, such as a lack of transparency, controllability, and user agency [Amershi et al., 2019, Buschek et al., 2021, Huang et al., 2020a]. Through Chaining, we aim to address these user-centered concerns.

In a collaboration, AI can play various roles, such as casual creators that encourage exploration [Davis et al., 2016] or assistants that compensate for human weaknesses [Levy et al., 2021, Wu et al., 2021]. For example, Gero *et al.* [Gero and Chilton, 2019] showed that generators could serve as cognitive offloading tools so that humans could focus their attention where it is needed most, a core motivation that we also share. Cai *et al.* [Cai et al., 2019] investigated how a medical AI can assist with doctors' decision-making process during prostate cancer diagnosis, by helping them compare and contrast similar images. Most of these studies, however, use task-specific models, and therefore limit observations to human interaction with AI that primarily serves one function, or in one domain (*e.g.*, writing, medicine, music, etc.). DuetDraw [Oh et al., 2018] may be an exception to this, as it uses several models, each of which supports a different co-drawing functionality. Rather than training multiple models for different tasks, or using a single model for a single type of task, our work explores how a *single* large language model (with inherently customizable capabilities) can support humans in *a variety of sub-tasks*. Finally, the closest work to ours might be online interfaces for users to interactively create prompts¹, or interfaces enabling users to perform natural language programming of code using a large language model [Jiang et al., 2021]. These systems used prompt engineering to create a set of programming-related functionality for users. While this prior work focused on single prompts, our work looks at how Chaining *multiple* prompts can address a much wider range of human tasks, and evaluate its effects on user experience.

¹<https://gpt3demo.com/apps/openai-gpt-3-playground>

6.2.3 Workflows in Crowdsourcing

Though less prevalent in human-AI collaboration, the concept of Chaining is inspired by concepts of “pipelining” and “microtasking,” which have long been used in crowdsourcing [Cai et al., 2016, Teevan et al., 2016]. In crowdsourcing, requesters break down complex tasks into pieces that can be performed independently, then combined [Chilton et al., 2013, Kim et al., 2017, Law and Zhang, 2011, Retelny et al., 2017]. Previous research shows that decomposed tasks allow the completion process to become more structured [Chilton et al., 2016] and more resilient to interruptions [Cheng et al., 2015], something we also witness in our user study. The goal of crowd workflows is typically to address and safeguard against the limitations of a typical worker. For example, Bernstein *et al.* [Bernstein et al., 2010] ensured text editing quality through a Find-Fix-Verify workflow, which modulates the scope of sub-tasks to reduce variance of crowdworker effort. Meanwhile, Context Trees [Verroios and Bernstein, 2014] hierarchically summarize and trim the otherwise overwhelming global contexts, making them compact enough for a single worker to digest.

Our Chaining approach also aims to address pitfalls of a single LLM pass, but the pitfalls are somewhat distinct. While crowdsourcing focuses more on cognitive load and task duration — factors that can affect the performance of human workers [Kulkarni et al., 2011] — for LLMs with intensive computing power, their limitations err towards a lack of reasoning abilities, high variance of prompt effectiveness, and exposure bias. A thorough analysis of these AI issues is needed for constructing and chaining LLM steps, which we illustrate in Section 6.3.1, and address through the design of primitive operations in Table C.1. Through user studies (Section 6.5) and case studies (Section 6.6), we demonstrate that Chaining can effectively address these issues. Finally, our work also shares challenges found in crowdsourcing workflows, such as handling cascading errors that affect later stages [Kittur et al., 2011], staged crash-and-rerun [Little et al., 2010], all of which we take into consideration in the design of the Chaining structure. Beyond this, we advance the field by examining how core features of Chaining (*e.g.*, cascading effects, parallel paths) are used not only to accomplish tasks, but also to aid in increasing the transparency and debuggability of AI.

(a) Validate and categorize the input		(c) Re-organize the input	
Def.	Classification: Assign the input to categories. Most useful for branching logic and validation.	Def.	Info. Extraction: Extract information from the context.
Ex.	Classify if <u>the question is answerable</u> . question: What is the square root of banana? is answerable (Yes/No): <u>No</u>	Ex.	Given <u>the text</u> , extract <u>airport codes per city</u> . text: I want to fly from Los Angeles to Miami. airport codes: <u>LAX, MIA</u>
<hr/>		<hr/>	
(b) Gather additional information from the LLM		Def.	Rewriting: 1-1 mapping that changes the input to more machine-readable formats (e.g., JSON to natural language).
Def.	Factual Query: Ask the model for a fact.	Ex.	Rewrite <u>the first-person text</u> into <u>third-person</u> . first-person text: I decided to make a movie third-person text: <u>He decided to make a movie.</u>
Ex.	Given <u>the US state</u> , find <u>the population</u> . US state: Washington Population: <u>7.6 million</u>	Def.	Split Points: 1-N mapping that is particularly useful for splitting contexts.
Def.	Generation: Ask the model to do some creative “hallucination” on the input.	Ex.	Split <u>the feedback paragraph</u> into a list of <u>Alex’s presentation problems</u> . Feedback: Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents. Alex’s problems: 1) <u>Too much text</u> 2) <u>No clear structure</u> 3) <u>Does not engage with audience</u>
Ex.	Given <u>the topic</u> , create a <u>two-sentence horror story</u> . topic: Breakfast two-sentence horror story: <u>He always stops crying when I pour the milk on his cereal. I just have to remember not to let him see his face on the carton.</u>	Def.	Compose Points: N-1 mapping, the reverse operation of decomposition; merge multiple results back together.
Def.	Ideation: Ask the model for a list of ideas or examples.	Ex.	Write one <u>friendly paragraph</u> to cover all <u>the problems</u> and <u>suggestions for improvement</u> . Alex’s problems: 1) Too much text; 2) No... Suggestions: 1) More images on the slides;... Review: Your presentation was interesting! However, I noticed that you have a lot of...
Ex.	Given <u>Alex’s presentation problem</u> , the following is a list of <u>suggestions</u> . Alex’s problem: Too much text Suggestions for improvements: 1) <u>Use more graphics</u> 2) <u>Use bullet points</u>	<hr/>	

Table 6.1: We curate eight primitive operations that may be adequately handled by a single LLM run. Grouped according to their intended objectives, these operations can help address the LLM challenges detailed in Section 6.3.1. Along with the definitions, we provide examples of prompts that enact these operations, with the underlined text being the LLM output given the preceding prompt. The examples for *Ideation*, *Split* and *Compose* points are replicas of steps in Figure 6.1. The full implementations (with the parameters in Figure 6.6) are in Appendix C.4.

6.3 Chaining LLMs

Despite the impressive capabilities of LLMs, there may be contexts in which LLM performance would suffer, such as if the data is formatted sub-optimally, if there is extraneous data in the input, if the task inherently demands solving multiple sub-parts, or if the user is asking the model to perform several tasks at once. Meanwhile, LLMs may perform *highly targeted* tasks well. By narrowing the scope and context of an LLM operation, for example, LLMs may themselves be useful for addressing some of their own challenges (e.g., removing extraneous data, splitting problems into

sub-parts, etc.). Thus, we hypothesize that decomposing a problem into smaller, highly targeted tasks is likely to increase model performance on those sub-tasks, and by extension, the overarching task.

We define *Chaining* as the process of breaking up complex tasks into smaller steps, where each step can be completed by an independent run of an LLM, and where the output of one or more steps is used as input for the next. To identify tasks that are most likely to benefit from Chaining, we first surveyed existing language modeling literature, and summarized **common challenges** LLMs face. As described in Section 6.3.1, these challenges are caused by the underlying modeling structure *shared* by the mainstream LLMs, including but not limited to GPT-3, Jurassic-1, and the internal LLM used in Section 6.5 and 6.6. Then, to identify promising sub-tasks that could be used as building blocks, we surveyed existing online demos of LLMs, and curated a list of **primitive LLM operations**, which may help overcome those challenges by scoping the inputs/outputs to be more amenable to what an LLM can handle.

6.3.1 LLM Challenges & Primitive Operations

Existing literature exposes three main challenges that LLMs face:

C.1 LLMs lack multi-step reasoning capabilities. Because LLMs are designed to grasp the *form* of language, rather than the *meaning* [Bender and Koller, 2020], they can struggle on tasks like sequential arithmetic problems, multi-hop question answering, recognizing and comparing sentences, or those that require branching logic [Betz et al., 2021, Floridi and Chiriatti, 2020, Wang et al., 2021a, Branwen, 2020, Wei et al., 2022].

C.2 LLMs suffer from *exposure bias* [Tan et al., 2021, Ranzato et al., 2016]. Because LLMs generate text sequentially in an autoregressive manner (the tokens generated by the models are themselves used to predict the next word), errors or imperfections from previous runs can accumulate. Thus, LLMs are less likely to perform well when generating long bodies of text. Exposure bias can also cause LLMs to produce redundant content, in some severe cases repeating the same phrase over and over again [Holtzman et al., 2020, Welleck et al., 2020]. As a result, they struggle to generate text with diverse themes or arguments (*e.g.*, suggestions for

all three problems in the peer review example in Figure 6.1).

C.3 LLMs are sensitive to input prompts. They tend to favor certain prompt formats, paraphrases [O’Connor and Andreas, 2021, Lu et al., 2021], or even certain information in the input. For example, prompts that are unnatural relative to the typical text distribution tend to be less efficient [Branwen, 2020], while nouns and verbs are more important than adjectives and function words [O’Connor and Andreas, 2021].

These challenges tend to stem from tasks being too broad. Yet, as discussed above, LLMs may be able to perform certain tasks well if they are highly targeted, with narrower contexts. Hence, with these challenges in mind, we reviewed 73 existing demos based on an extensive search of official LLM websites, social media, and published case studies (these are enumerated in Table C.1, Appendix C.1) to identify promising LLM capabilities that may help scope the inputs/outputs, culminating in a set of primitive operations. Note that the operations we identified may not be exhaustive, but rather represent an interesting range for study, with a variety of operations addressing each LLM challenge. Pilot studies — as well as use cases we present later — suggested these were a reasonable set to pursue. Full details of our methodology can be found in Appendix C.1.

Table 6.1 shows how the derived operations fall into three categories and can address the aforementioned challenges. First, as LLMs may have difficulty applying common sense reasoning or complex inference to nuanced problems (C.1), the *Classification* operation can act as a **validation check** or triage, before more steps are carried out (Table 6.1a). For example, a chatbot may need to first classify the type of question a user is asking before providing adequate responses. Second, to alleviate exposure bias (C.2, the inability to generate long and diverse text), some operations can be used to **query small chunks of new content** (Table 6.1b), so as to gradually build up the generation diversity and length. Three ways to get new content include *querying facts*, *generating hallucinations*, and *ideating* lists of items. For example, in the peer review rewriting scenario (Figure 6.1B), the separate *Ideation* step per problem prevents suggestions for one criticism from being influenced by the other two criticisms. Finally, because LLMs may struggle with certain input prompt types, **reorganizing the prompt** could be helpful when its original form is convoluted. *Rewriting* and *Compose points* transform input into more parsable forms, *Information Extraction*

elicits concise information (C.3), and *Split points* splits text into smaller and more manageable units (C.1)—all are summarized in Table 6.1c. As we will see in a case study (Section 6.6.1), translating JSON-formatted specifications to natural language descriptions helps LLMs parse the embedded information.

Chaining and its operations also have some parallels to crowdsourcing workflows. However, whereas sub-tasks in crowdsourcing are assumed to be feasible for a human worker (reviewed in Section 6.2.3), LLMs are more restricted in terms of tasks they can perform reliably, and thus the primitive operations presented are more scoped and granular. For example, Kittur *et al.* [Kittur et al., 2011]’s Partition-Map-Reduce workflow uses *Split* and *Compose Points* operations (in Figure 6.1B), but does not indicate specifically how to transform the text (*Ideation*), though it also targets collaborative writing.

6.3.2 Designing Operations for LLM Chaining

An LLM Chain consists of multiple **steps**. Each step is defined by an **LLM operation**, which takes in input data and produces output data (which we call **data layers**). For example, the *Split point* operation in Figure 6.1 takes in the ● initial feedback for Alex as input, and produces a list of ● presentation problems (“too much text”, “no clear structure”, etc.) as output. LLM Chains are constructed by connecting these steps through shared data layers. In the same example above, the *Ideation* operation comes after the *Split points* operation, taking a (previously generated) ● problem as input and producing ● suggestions for improvements as output.

Each step of an LLM (an operation and its data layers) is accomplished through a natural language **prompt**. While prompts are task-dependent, they can have some task-agnostic properties. For example, the prompt for the *Classification* operation would likely contain the verb “classify”, regardless of what is being classified. These keywords help set an LLM operation’s scope and expectations [O’Connor and Andreas, 2021]. We aim to abstract these task-agnostic properties into default parameters for each operation (Figure 6.2A), so as to provide consistent starting points for interacting with LLM Chains across use cases. Using the *Ideation* operation as an example, we show how we design these parameters to satisfy the following three requirements for chaining, and

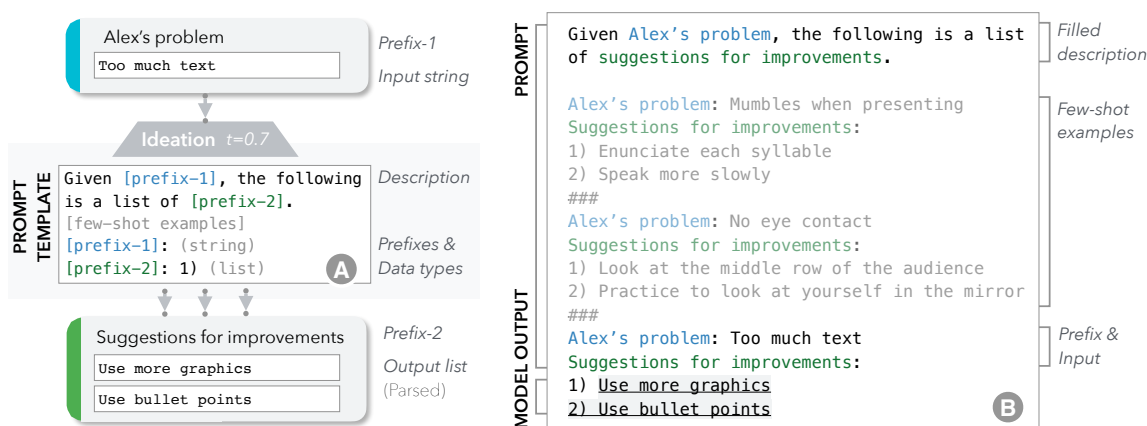


Figure 6.2: An example of how to create an LLM step using a **prompt template** (A), using the *Ideation* step of the peer review writing scenario (from Figure 6.1) as an example. For the peer review scenario, the *Ideation* operation takes in a problem (e.g., too much text) as input, and produces suggestions for improvement as output, but the prompt template allows the *Ideation* operation to take in any custom inputs and outputs. The template includes placeholders for the input (**prefix-1**), output (**prefix-2**), and (optional) few-shot examples. (B) shows the actual prompt after filling in the placeholders in the prompt template.

how they help to build the *Ideation* prompt shown in Table 6.1 and Figure 6.2B.

Operations need to invoke the desired functionalities, through prompt design. To date, the most common patterns for prompting are either *zero-shot* or *few-shot* prompts, depending on how many demonstrating examples are provided in the prompt [Brown et al., 2020]. Zero-shot prompts directly describe what ought to happen in a task: e.g., we can enact *Ideation* with a *task description* prompt “Given Alex’s presentation problem, the following is a list of suggestions.” In contrast, *few-shot* prompts show the LLM what pattern to follow by feeding it examples of the desired input and output data: “Problem: mumbles when presenting, Suggestion: enunciate each syllable, Problem: too much text, Suggestion:” (full prompt in Figure 6.2B). Given these prompts, the LLM might produce a reasonable suggestion, e.g., “use more graphics on the slides.” Zero-shot prompts can also be easily transformed into few-shot prompts, by appending examples to the initial zero-shot task description. In either case, prompts commonly include meaningful names as **prefixes** (“Problem:” and “Suggestion:”) to demarcate structure, which helps re-emphasize the desired

intent [Vaswani et al., 2017]. Following this convention, we build our prompts to include *task descriptions* followed by *prefixes*. Aside from the prompt itself, we also associate with each LLM operation a default **temperature setting**: a model parameter that influences the randomness of the LLM generation. For instance, creative operations like *Ideation* benefit from a higher temperature ($t=0.7$) than more factual or deterministic tasks like *Classification* ($t=0.0$) [OpenAI, 2021].

Operations should be able to take custom data layers as inputs and outputs. Though our walkthrough example takes in “Alex’s presentation problem” and generates “Suggestions”, in theory an operation should be able to handle any custom data layers. We thus create **prompt templates** to support a wide range of scenarios, with placeholders for input and output data. The template allows us to build LLM steps simply by filling in the placeholders with definitions on data layers, as demonstrated in Figure 6.2. In particular, we include key verbs and nouns [O’Connor and Andreas, 2021] in the template, to best reflect the operation objective (*e.g.*, “a list of” for *Ideation*, “classify” for *Classification*). The template also accepts optional few-shot examples. We can build the few-shot prompt in Figure 6.2B if we provide those pairs of problems and suggestions, or default to just the zero-shot version in Table 6.1 when examples are not readily available. Though we provide this as one example of a prompt template, we do not claim it to be exhaustive as there may be other equally effective ones.

Operations should handle parsing of the expected input/ output data types. Different data layers may take on different data types. For example, the Split step (Figure 6.1 b_1) produces a *list* of problems, but only a *single* problem is the input to each subsequent Ideation step (b_2). To handle different formats in different steps, in each operation’s definition, we define the required data types per operation (*e.g.* “list” in Figure 6.2 for *Ideation*), along with the corresponding parsing necessary to produce the expected data type (*e.g.*, split each row of the numbered list into an item).

Empirically, we find these defaults to work reasonably well across domains (see later sections 6.5 and 6.6). Still, we note that our defaults here are just one example of possible operation implementations; in our review of existing demos, there appeared to be many diverse prompting strategies even for the same task. We hope the prompt templates provided here may serve as a starting point for Chain designers or users to modify. In the next section, we demonstrate how these designs serve

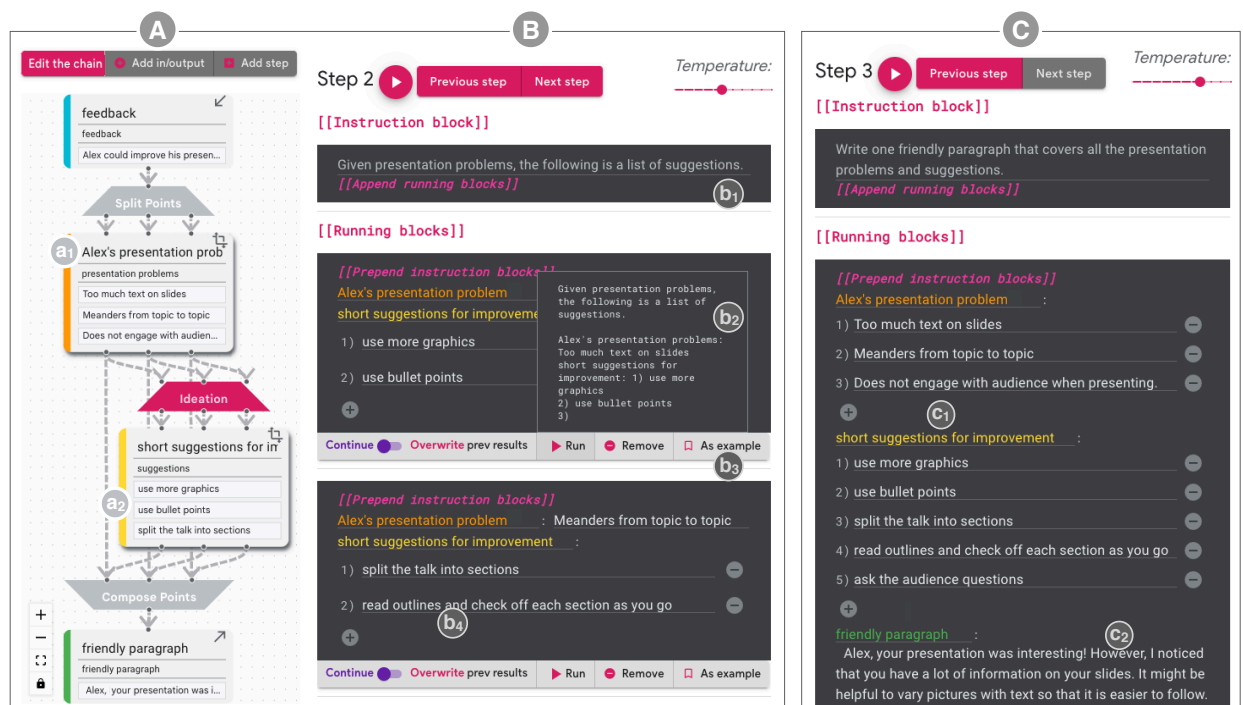


Figure 6.3: An overview of the interface, reflecting the peer review rewriting example in Figure 6.1. It consists of (A) a *Chain view* that depicts the high level Chaining structure, and (B/C) a *Step view* that allows for refining and executing each LLM step. The interface facilitates tracking the progress of the LLM Chain. For example, when moving from step 2: *Ideation* (B) to step 3: *Compose Points* (C), the previously generated presentation problems and suggestions become inputs for the final paragraph. A demonstration is available at <https://youtu.be/QFS-1EW1vMM>.

as the underlying data structure for interactive Chain execution by end-users.

6.4 Interactive User Interface

We designed an interface that helps users execute and customize LLM Chains interactively.

6.4.1 Design Rationales

Over the course of several weeks, we designed and iterated on the prototype with feedback from four pilot users (software engineers and designers who have experience designing LLM prompts),

producing three design rationales for the final interface.

R.1 Visually reflect the underlying Chaining structure. In early prototypes, we explained the Chain structure using a static slide deck that highlighted the *data* produced at each step (*e.g.*, problems, suggestions for improvement, and final paragraph in Figure 6.1). In reaction, users expressed a desire to understand the *operations* taken at each step to arrive at these data layers (split points, ideation, compose points), and wanted to visually track progress through the Chain. To achieve this, we designed the interface to reflect not only the data layers, but also the LLM details within each step.

R.2 Provide controls at different granularities. Pilot users favored flexible controls. We observed users frequently making local fixes on intermediate data points that flow between LLM steps, and therefore designed the UI to allow in-place editing, without explicitly requiring a switch to editing mode. Some users also voiced an interest in iterating on alternative Chaining structures (“*Can I change this step with...*”). We therefore conclude that the interface should support modification of LLM Chains both locally (*e.g.*, changing one task description or intermediate model output) and globally (*e.g.*, changing how the steps are connected). Because global changes have more impactful consequences (they may overwrite the underlying Chain structure), we designed the UI to require a switch to editing mode for this type of changes.

R.3 The structured controls should still reflect the natural language interaction supported by LLMs. In an early prototype, we formatted the data as structured tables with each data layer being a column, but received feedback from two users that making text edits in cells felt unnatural as they lost the sense of interacting with the model through natural language. To retain a natural interaction experience, we keep these structures as in-line text fields.

6.4.2 Interface Design and Implementation

We design the interface in Figure 6.3 following these design rationales above, which consists of two primary views: the *Chain view* (Figure 6.3A), and the *Step view* (Figure 6.3B/C).

The **Chain view** (Figure 6.3A) depicts the high level Chaining structure through a flow chart. It contains three primary visual cues that closely reflect the underlying design (R.1) described in Section 6.3.2. First, we use grey glyphs to represent *LLM operations*, with shapes indicating 1-1 (rectangle, for operations like *Rewriting* in Table 6.1), 1-N (trapezoid, *e.g.*, *Ideation* operation), and N-1 data mappings (inverted trapezoid, *e.g.*, *Compose points* operation). Clicking on these glyphs allows users to choose which step to zoom into (highlighted in pink), and the Step view would change in response. Then, we use rectangles with colored stripes to represent *data layers*. Users can preview their data entries through white rows (*e.g.*, Figure 6.3a₁ and a₂), which are updated after each LLM execution, and thus track Chain execution progress. Finally, we link these elements with dotted-line arrows to highlight which data output serves as the input to which step, and use the number of arrows going out of an operation to re-emphasize the data mappings (*e.g.*, multiple ● problems coming out from *Split points*, which is approximated with three lines, and a single ● paragraph out of *Compose points*).

On the right, the **Step view** (Figure 6.3B) allows users to explore each LLM step by interacting with inputs, outputs, and the underlying prompt structure. It is divided into an *instruction block* and several *running blocks* to handle *parallel paths*. Each of these parallel paths translates to a different LLM invocation; they share some common parts in their prompt strings, while having other parts being distinct from each other. We use the running blocks to hold the unique parts, and the instruction block to hold the shared sub-string is pre-pended to *all* running blocks, such that they are combined to form the full prompt. For example, Figure 6.3b₂ is the final prompt for the step that generations suggestions for the problem “too much text.” It starts with the content from the instruction block (*b*₁), and merges the text in the running block thereafter, ignoring the other parallel running blocks.

Every running block visually resembles a textarea with a number of editable text fields. It shows the prefix fields before colons (*e.g.*, ● Short suggestions for improvement, *c*₁) in the same color as the data layer rectangles, which helps users distinguish between data layers. It also includes text fields (*b*₄, *c*₂) for the model output for that step. The number of text fields (*e.g.*, 1 vs. N) are consistent with the data types defined for the primitive operation for that step.

This view also handles the per-step execution. Users can click the small “run” button to execute each running block individually. Alternatively, users can use the Play button on the top to run all the parallel blocks at once and compare their results. To improve natural language interaction transparency (R.3), running a block also triggers a preview of the final prompt text (b_2). The output is then parsed and added to the corresponding field (b_4, c_2) for users to further iterate on.

Interactions and controls. Notably, there are three levels of control available with this interface (R.2), from local customization of prompts to global modification of the LLM Chain structure, each with clear cues on its impact. First, users can customize the prompt for a particular step, *e.g.*, by changing its task descriptions. Since the customization only applies to the current step, all other views remain unchanged. Second, users can customize the model output for that step by adding, deleting, or editing content (*e.g.*, editing “read outlines” to *emphasize main points* in b_4), or rename data layers (*e.g.*, rephrasing “Alex’s presentation problems” as “Criticisms of Alex” in a_1). These changes impact both the current step in focus as well as other steps involving the shared data layers (*e.g.*, *Compose Points* takes in both the “problems” and the “suggestion” layer), and thus they can be changed either in the colored rectangles in the Chain view, or through text fields in the Step view. Finally, users can more aggressively modify the Chaining structure itself by adding, removing and rewiring operations or data layers in the Chain view through intuitive visual programming (R.3). The change would then cause the entire Chain to re-render, with the defaults (*e.g.*, temperature, instructions) refreshed.

6.5 User Study

To understand how Chaining affects the user experience of accomplishing tasks with LLMs, we conducted a within-subject user study comparing Chaining with a state-of-the-art baseline interface, on two user tasks.

6.5.1 Study Design

Underlying LLM. All of our experiments — including our baseline interface introduced below, and *each step of the Chaining interface* rely on exactly the same underlying LLM called LaMDA [Thoppilan et al., 2022]²: a 137 billion parameter, general-purpose language model. This model is roughly equivalent to the GPT-3 model in terms of size and capability: it is trained with more than 1.5T words of text data, in an auto-regressive manner using a decoder-only Transformer structure which is most useful for text generation. It has comparable performances with GPT-3 on a variety of tasks, and behaves similarly in its ability to follow prompts. Note that we only use this model to represent the recent class of LLMs; Essentially, the chaining interface is model agnostic, and is compatible with any LLM that has in-context learning capability.

Systems. We compared Chaining with Sandbox, an interface that looks aesthetically similar to the Chaining interface, but without the Chaining functionality. We based the Sandbox interaction on GPT-3 playground,³ the standard online interface for LLMs. It presents a single textbox with a run button, which allows the user to enter the text prompt, run the model on that prompt, and then view the model result in the same textbox, with the ability to edit that result and then continue to iterate. Like the Chaining interface, the Sandbox also allows users to adjust the temperature setting through a knob.

Tasks. We conducted the study using two tasks: peer review writing, and personalized flashcard creation, as they reflect different types of challenges (as explained below), and are both commonly used in user-centered task scenarios [Cai et al., 2014, Edge et al., 2011, Cambre et al., 2018]. In the peer review writing task (“**Review**,” our walk-through scenario), the user is given a paragraph (the same as in Figure 6.1) outlining three different problems in an imaginary person’s presentation style, and their task is to write a friendly paragraph with 1-3 suggestions for each problem. In flashcard creation (“**Flashcard**”), participants were asked to create at least ten English-French sentence pairs they could use while traveling in Paris, and to make them as diverse as possible while

²We used a non-dialog version of the model.

³<https://gpt3demo.com/apps/openai-gpt-3-playground>

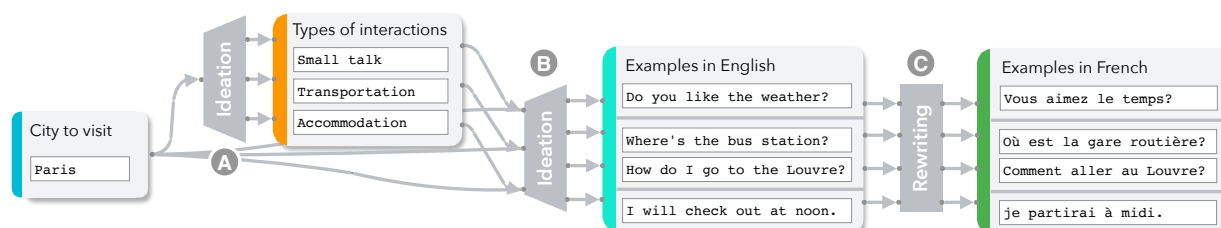


Figure 6.4: The LLM Chain for flashcard creation, with: (A) An **Ideation** step that brainstorms the ● types of interactions that we might encounter when ● visiting a given city (Paris), (B) Another **Ideation** step that creates a list of ● English examples for each ● interaction type, and (C) A **Rewriting** step that translates each ● English example into ● French.

being personalized to their own travel goals.

Though both tasks are possible when using an LLM without any LLM Chains, they present different types of challenges which could potentially be improved through Chaining. The *Review* task implicitly involves multi-step reasoning (Challenge C.1 in Section 6.3): to create a thorough and constructive review, one needs to identify each problem, provide suggestions per problem, and compose all the suggestions into one paragraph. The *Flashcard* task, on the other hand, exposes the challenge of having sufficient diversity in light of LLM exposure bias (C.2). In the Chaining condition, we built a default Chain for each task. The Chain for *Review* in Figure 6.1 reflects the three aforementioned steps (as explained before); the Chain for *Flashcard* (see Figure 6.4) sources additional content from the LLM like ● types of interactions in a trip, which can help the user diversify the flashcards.

Study procedure. Before the study, participants completed a 30-minute tutorial that summarized the concept of LLMs and demonstrated how both *Sandbox* and *Chaining* work.⁴ They were told upfront that both systems rely on the same underlying LLM. Then, in an hour-long study, participants performed a randomly selected task (*Flashcard* or *Review*), once with each interface (*Sandbox* and *Chaining*), whose orders were counterbalanced. We first briefed participants on the task, and then asked them to accomplish it with LLM’s help in each interface until they were satisfied with the final

⁴We took inspiration from the OpenAI: <https://beta.openai.com/docs/introduction/prompt-design-101>; the task used for tutorial is in Appendix C.2.3.

results, or until they reached 25 minutes. Since LLM Chains came with automatically generated prompts (by filling in the templates), we similarly offered several default prompts for Sandbox that we knew to work reasonably, so that both interfaces had a fair starting point for prompt engineering (detailed in Appendix C.2). We encouraged participants to think aloud and describe their actions as they completed the task.

In the *Chaining* condition, participants were asked to first stick to the default Chain so that we could make consistent observations across participants in terms of how they *use* Chains. In the process, they could modify any other aspect (*e.g.*, the prompt, the intermediate model outputs, etc.) At the end, we gave participants the option to *modify* the default Chain, so that we could observe how they would expect the LLM to assist them beyond the default design. Finally, participants completed an exit survey and a semi-structured interview. They rated their experience using each interface along various dimensions. These dimensions were chosen to reflect the effectiveness of the human-AI collaboration (*e.g.*, support for their thought process, quality of the final result), and core user-centered challenges in human-AI systems [Amershi et al., 2019, Buschek et al., 2021, Huang et al., 2020a] (*e.g.*, transparency, controllability, and sense of collaboration). They also verbally compared their impressions of the two interfaces, and envisioned possible use cases for them.

Collected data. We collected and analyzed three sets of data. First, to assess participants’ **self-perceived experience**, we used a standard seven-point Likert Scale [Likert, 1932] to collect all ratings from the exit survey, with one being “Strongly disagree” and seven being “Strongly agree” with the statement in question (*e.g.*, for system Transparency: “The system is transparent about how it arrives at its final result”). Detailed survey questions are listed in Appendix C.2.1.

We also observed and recorded their entire task completion sessions, and later transcribed their comments and experience for qualitative analysis. Second, to quantify their **interaction mechanisms and behaviors**, we logged their interactions with the two interfaces. We were particularly interested in how participants reacted and iterated on model outputs, so we sorted their interactions with text fields by: (1) whether participants mainly relied on running the model again to get a different result (*Consecutive run*), or if they also edited the prompt in between (*Edited*); and (2) when they

edited the prompt, how dependent it was on the existing model generation: whether they closely CURATED and refined the model outputs, loosely interacted around them by CREATING completely new content, or tried again by UNDOING the outputs. The detailed categorization criteria is in Appendix C.2.2. Third, to assess the **task outcome**, we logged the final reviews and flashcards participants created. Blinded to the condition, two non-participants performed anonymous, paired comparisons on results from each participant in Sandbox and Chaining, choosing the result that satisfied the task goals the best.

Participants. We recruited 20 participants using email lists that reach a wide range of practitioners (*e.g.*, UX designers, linguists, data analysts) at a large software company. Eight participants were 26-35 years old, eight aged 36-45, two aged 46-55, one 56-65, and one 18–26. As there is an initial learning curve associated with LLM capability, we required that participants had at least seen an LLM example before. Among those we recruited, half of the participants had no prompting experience but had seen online demos powered by LLM models, whereas the other half had some basic experience using default text prompts. Further, as the goal of Chaining is to use LLMs to assist with human tasks, we sought to recruit potential *users* of ML/LLM who would benefit from interacting with the models, rather than ML model experts or *creators*. Thus, our participants included technically knowledgeable but non-ML software engineers, linguists, UX designers, and data analysts who worked in a wide range of domains (*e.g.*, health, privacy, cloud storage, etc.). Each participant spent approximately 90 minutes total in our study, and received a \$40 gift certificate for their time.

6.5.2 Quantitative Results: Increased Transparency & Control, and Higher-quality Task Outcome

All the participants were able to complete the tasks in both systems within the given time: they spent 12.4 ± 4.0 minutes in Sandbox, and 14.6 ± 5.4 in Chaining. Student's t-test did not show any significant difference between their completion time ($t = -1.1, p = .278$). In analyzing subjective ratings from participants, the logged clickstreams, as well as the final generated results, we found:

First, *Chaining led to improved user experience in human-AI interactions*. We performed the non-parametric Wilcoxon signed-rank test to compare users' nominal Likert Scale ratings



Figure 6.5: Participants’ ratings in the form of seven-point Likert scale questions (details in Appendix C.2.1), with 95% confidence intervals. Using Chaining, participants felt they produced results that better *matched the task goals*, and that the system helped them *think through* the task. They also found Chaining more *transparent*, *controllable*, and *collaborative*.

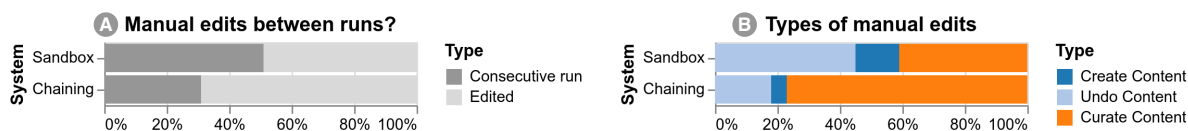


Figure 6.6: Distribution (based on the logged interactions) of how participants interacted with the prompts and model outputs, with and without chaining. (A) They made more *edits* in Chaining (compared to just repeatedly running the model), and (B) They tended to *curate* model outputs, rather than either deleting (*undoing*) them entirely or manually *creating* new content.

and, as shown in Figure 6.5, participants felt that Chaining helped them **think through the task better** (Chaining 6.0 ± 1.4 vs. Sandbox 3.6 ± 1.3 , $z = 0$, $p < .001$), and **gave them more control** (6.2 ± 0.9 vs. 4.5 ± 1.3 , $z = 3.0$, $p < .001$). They also rated Chaining as being more **collaborative** (5.7 ± 1.3 vs. 4.6 ± 1.6 , $z = 25$, $p = .04$) and **transparent** (5.4 ± 1.3 vs. 3.8 ± 1.8 , $z = 9.0$, $p = .002$).

Second, *Chaining shifted the types of edits participants made while interacting with the LLM*. In Chaining, participants were more likely to make manual interventions, whereas in Sandbox, they often re-ran the model (without changing the prompt) — akin to “rolling the dice again” in an attempt to get better output. As shown in Figure 6.6A, this tendency to perform consecutive runs without altering anything from the previous run occurred 51% of the time on average in Sandbox

and 36% in Chaining. Student's t-test shows the difference is significant: $t = 3.5, p = .001$.⁵

The manual edits made were also finer-grained in Chaining than in Sandbox (Figure 6.6B). In Sandbox, people largely focused on either completely UNDO output and rerunning the model (45% of the time on average), or manually CREATING their own content as input to the model (14%). They only CURATED or modified existing text 41% of the time. On the other hand, in Chaining people performed CURATION 77% of the time, only doing UNDO and CREATE 18% and 5% of the time, respectively. The shift to CURATION is significant, according to Student's t-test ($t = -6.75, p < .001$).

As a result, *Chaining led to higher-quality generations that met the task goal*. The two independent raters consistently preferred Chaining results 85% and 80% of the time, respectively. The results also matched participants' own judgements in Figure 6.5 (see *Match goal*) — they preferred their own final results from Chaining (6.0 ± 0.9) to the Sandbox results (5.0 ± 1.1 , Wilcoxon signed-rank test, $z = 11.0, p = .002$).

Aside from *using Chaining*, many participants were also able to iterate on and customize the underlying Chaining structure. While five of them preferred the default Chains provided and didn't want to change them, the remaining 15 people were able to identify parts they found lacking and suggested at least one change. 11 of them successfully implemented and executed one of their own solutions.

6.5.3 Qualitative results: Chaining as Guardrails and Operation Manuals

Through analyses of the transcribed think-aloud comments and semi-structured interviews, we further unpack the reasons behind the quantitative differences. Since we asked participants to explain their Likert Scale ratings, their interview responses naturally map to dimensions in Figure 6.5 like transparency, collaboration, etc. One author further sorted their think-aloud comments into the categories. Three researchers then conducted thematic analysis, examining relationships between categories and iteratively converging on a set of higher-level themes. In general, Chaining helped

⁵The clickstreams fall into the continuous range of 0%–100%, and follows a normal distribution according to a D'Agostino-Pearson Test (e.g., $p = 0.58$ for the ratio of consecutive runs).

support human-LLM interaction by serving as (1) a guardrail that helped users stay on track towards the task goal (Section 6.5.3 and 6.5.3); and (2) an “operation manual” that implicitly explained how to use LLMs for less obvious objectives, and that provided channels for users to intervene (Section 6.5.3, 6.5.3 and 6.5.3). In the following sections, we present key themes on how Chaining improved the human-AI experience, as well as some additional challenges brought on by Chaining.

Chaining helped users more fully capitalize on the model’s latent capabilities.

In *Sandbox*, participants tended to use the LLM for a single purpose, under-utilizing the model’s full potential in supporting various kinds of tasks. Four out of ten people in the *Flashcard* task only used the model as a translator in *Sandbox*, even though they were provided with default prompts that demonstrated how to generate English sentences using the model. In the *Review* task, even though nearly everyone (nine out of ten) used a two-step process of generating suggestions prior to merging them into the full paragraph (see the two-step prompt template in Appendix C.2.5), three people only relied on the LLM to generate suggestions, and then manually merged them into the paragraph themselves, without LLM input.

There may be two reasons for these behaviors. First, *Sandbox* naturally affords single-operation interactions. Given this, it is not surprising that users would gravitate toward using the model only for a part of the task that seemed most likely to yield promising results given the *status-quo* applications of machine learning (*e.g.*, translation), overlooking others that may seem less likely to succeed (*e.g.*, merging text into a paragraph). Indeed, some participants were unaware of less obvious sub-tasks (P4: “*this is just a simple translation task*” in *Flashcard*). Second, the friction of juggling multiple sub-tasks in *Sandbox* deterred some users from doing so. Even participants who became aware of the Chaining structure (from getting the Chaining condition first in their study condition order) struggled to replicate it using a single prompt. For example, P2 attempted to tackle both sub-tasks (generating diverse English sentences, and translating to French) simultaneously with a single prompt instruction: “Given the previous English sentence, translate it to French. Generate further English sentences relevant to travel in Paris.” However, because the instruction was too nuanced for the model to follow, they eventually resorted to manually creating their own English

sentences.

Ultimately, this inability to fully utilize the model led to lower quality final results in Sandbox. For example, the flashcards had less topical diversity (P4: “*I had limited diversity myself*”) because the *Ideation* step in Figure 6.4A was rarely ever leveraged. As a byproduct of the inadequate support, participants also found collaboration in Sandbox to be shallow (P5: “*I’m doing all the specific work [creating English sentences] and it’s just doing its one thing [translation]*”). In contrast, **Chaining** allowed users to leverage the model in multiple ways. Seven participants particularly liked that they could accomplish multiple goals through the Chain, *i.e.*, acquiring model-powered diversity in the *Ideation* step, while maintaining translation correctness in the *Rewriting* step. This additional support may have contributed to participants shifting from creation (manually creating text from scratch) to curation (modifying model outputs) as shown in Quantitative Results (Figure 6.6B). Quoting P5, “*I didn’t need to give it as much, but it was giving me a lot.*”

LLMs’ diverse primitive operations and capabilities also led participants to consider other ways the model might be helpful. For example, when asked to modify the Chaining structure itself, P1 in *Flashcard* swapped the *Ideation* step (which generated ● types of interactions) with a *Generation* step to produce ● a journal of my one day trip, so the model could “*think about what conversations can happen across my day trip*” and provide “*less generic context suggestions.*” The operations became inspirational here. P12 and P20 in *Review* both added a *Classification* step to determine if the paragraph is in the right voice or if a suggestion is actionable, only once they realized the classification operation existed.

The ability to isolate interventions and save progress enhanced controllability of LLM

Because each step of a Chain involves a separate run of the model, Chaining allowed users to control certain aspects of each sub-task *independent of* others. Four *Flashcard* participants in Chaining noticed that the desired model randomness should vary per sub-task, and tuned the temperature settings accordingly: they increased the temperatures in *Ideation* steps to broaden the diversity and creativity of model responses (Figure 6.4A and B), and lowered it for *Rewriting* to increase the chances of getting correct model output (Figure 6.4C). However, none of them did so in the

Sandbox condition (e.g., P5: “*I realized my temperature was always high in sandbox. I should have had it low at translation, and high when I ask the model for English sentences.*”) Many Review participants also liked iterating on each of the presentation problems individually (e.g., “To much text on slides” vs. “No clear structure”) without affecting the others.

This well-scoped impact of interventions may explain why participants felt more motivated and comfortable making manual edits in Chaining (Figure 6.6A). Nine people felt more compelled to enact controls on sub-tasks, knowing that they did not have to worry about unintended effects on other parts.

Four of them further noted that this clean separation would be tedious (if not impossible) in Sandbox, hence the differences in the perceived controllability in Figure 6.5. For example, P13 in Review attempted to replicate the exact same Chain in Sandbox. They manually divided the original paragraph into three problems, then asked the model for suggestions for each, and to compose the final paragraph. However, rather than storing suggestions externally and starting fresh for each problem, they simply stacked them together in a single prompt: “Original paragraph:...; Problem: too much text; Suggestions: 1)...; Problem: Split...” The resulting long and intertwined text became overwhelming: “*I was very nervous to edit anything, because I didn’t know how that was going to impact the end task goals.*”

Beyond staged interventions, staged outputs also provided participants with the opportunity to evaluate and improve individual components irrespective of previous failure [Nushi et al., 2017]. Three participants praised the ability to “freeze” their preferred intermediate data points: “*I reached some point of some progress in the middle of the Chain and if this works, then it’s fixed when I play with the next step. It doesn’t get lost — unlike the sandbox, where whenever I change something somewhere the result will be completely different*” (P10). Their observations are also in line with the crash-and-rerun capability of crowdsourcing [Little et al., 2010], where local reruns are desirable without affecting previous stages.

Surfacing the Chaining structure increased transparency.

Chaining enriched system transparency, which helped participants better calibrate their expectations of the model. As each step of the Chain had a specific role (*Ideation*, *Rewriting*, etc.), they helped narrow the scope of the model’s intended functionality, making it easier for participants to understand what to expect from a model that might otherwise seem all-encompassing. Nine participants noted this benefit of calibrated expectations. For example, P6 commented that “*Chaining helped you speak the language. It lift[ed] up the hood and showed you the steps and what’s happening at different phrases,*” and P15 stated that “*having default settings like your templates gave me an idea of how it works.*” As elaborated in Section 6.5.3, having isolated steps, each with a reduced scope, also help users better anticipate the potential impact of their inputs, further increasing system transparency.

More globally, Chaining enabled users to develop a more accurate mental model of the LLM’s capabilities, by allowing them to tinker with sub-components in a modular and comparative manner. Users could, for example, compare parallel paths to deduce how the model would respond to alternative inputs. In the *Flashcard* task, P8 noticed during the *Ideation* step that the model generated more useful English sentences when the ● types of interactions was “accommodation,” compared to “topics related to public transportation.” This hinted at the model’s better performance when presented with a useful keyword. Modifying the order of LLM steps also enabled users to learn aspects of the model’s strengths and weaknesses. When customizing the Chaining structure, five participants tried adding another *Rewriting* step either after the final paragraph (at the end of the Chain), or on the individual presentation problems (early in the Chain). Though initially unaware that LLMs can suffer from exposure bias (see C.2), participants quickly discovered through this comparison that the model could more effectively modify *sentences* than *paragraphs*. This comparison was rare in *Sandbox*, as it was not obvious to participants that they could keep the LLM functionality but shorten the input.

Surfacing the Chaining structure increased debuggability.

The increased transparency in Chaining also gave users better debugging mechanisms. When the model output was inconsistent with user intent, participants were at a loss for what to try next in Sandbox. Because users could conceivably type and modify any natural language prompt in the text box, the scope for “debugging” was too expansive. P9 remarked that *“too much freedom can be a curse,”* while P7 felt like *“sitting down in front of the controls of an airplane, all the knobs are there but I don’t know what to do with them.”* Instead, Chaining exposed intermediate knobs that helped participants draw a more direct connection between observed model deficiencies, and possible remediation. P9 found it easier to debug by modifying the inputs and outputs for each step of the Chain, rather than merely re-running the model in Sandbox repeatedly, in the hopes of more promising model results (*“I had to constantly delete and rerun things.”*). This may explain why the frequency of UNDO actions was reduced in Chaining (Figure 6.6B).

Accordingly, three interesting debugging mechanisms emerged: First, the isolated steps in Chaining acted as AI “unit tests” that enabled users to pinpoint a seemingly global error to its local cause. For example, participants in *Flashcard* frequently removed topics irrelevant to traveling (*e.g.*, education), so that sub-optimal solutions would not be fed into subsequent steps. Second, the ability to create parallel paths and alternate step orders (elaborated in Section 6.5.3) enabled *comparative debugging*. Revisiting the case mentioned above, observing a higher-quality path (*e.g.*, using a simple keyword in the prompt like “accommodation”) helped participants infer how to improve prompts in other parts of the Chain (*e.g.*, changing “topics related to public transportation” to “public transportation.”)

Finally, the ability to propagate a change throughout the entire Chain gave users *immediate feedback on whether a fix was successful*, thereby shortening feedback and iteration cycles. For example, P3 renamed ● types of interactions with ● places where conversation might occur, so as to *“have flashcards grouped by happening at the airport, restaurant, while walking around streets.”* They were impressed by the changes propagating to the final results: *“you can just change a step without affecting other steps but then your final results are reshaped based on that. I didn’t think that*

was going to work that simply.” This combined ability to both isolate and propagate interventions was key to increasing AI debuggability.

Scoped objectives in sub-tasks served as guardrails against LLM-inspired tangents

One challenge that hindered participants’ performance on the tasks was LLMs’ randomness and creative surprises. The model would often produce outputs that were compelling in their own right, which in turn would derail people from the intended task. For example, P5 in *Flashcard* was intrigued by an LLM-generated English sentence, “That man is suspicious to me,” and started tricking the model into writing a story — “*I want to know what happened to the suspicious man!*” Five out of twenty people wandered from their task goal in *Sandbox* and began exploring tangents or attempting to “break” the model. They had to be reminded several times to get back on track. Participants later recalled their habit of drifting: “*I tried a lot of cool things, but it’s not the task I want to complete*” (P17).

Interestingly, we found *Chaining* acted as a safeguard against model-inspired tangents, not only because each step of the Chain defined a clear goal, but also because the interconnected data layers motivated participants to deliberately steer outputs of each step away from cascading errors (*e.g.*, incorrect problem extraction in the first step of Figure 6.1b₁ could lead to a poor final paragraph). In the *Ideation* steps, participants would even manually move model output around to make sure they fit the topic (P7: “*this isn’t really about asking for directions, I should put it in accommodation.*”) Ultimately, participants treated the entire task more carefully (see Figure 6.5, *think through*) — “*if I was trying to do it with speed, I might find the sandbox easier; but if I want to do it with precision, I prefer the Chaining structure.*” (P13).

Additional challenges

Chaining brought many benefits to human-AI collaboration, but it also presented several challenges. Nine participants noted that although they found the Chains to be transparent, rich, and educational, they were also more complex, with steeper learning curves. Moreover, while Chaining enabled

participants to zoom into sub-tasks in modular ways, it also occasionally made the larger picture more difficult to recall: Four participants had questions about *“how my particular change to this data entry will affect the final result”* in Chaining (P2), and commented that the end-to-end aspect of Sandbox enabled them to see the direct effects of their actions. These challenges may have been a side-effect of participants using pre-defined Chains, which may not necessarily reflect their own intuition of how they would have decomposed the task [Xie et al., 2017, Carroll and Olson, 1988]. Most people had a much more fluent experience with the Chains they modified — *“I liked creating my framework.”* (P13). Though beyond the scope of this chapter, this raises the question of how to support users in not just using Chains, but also authoring their own Chains, to improve user agency and intuitiveness of Chaining [Wu et al., 2022a].

Moreover, while Chaining provided better guardrails for staying on task, it may come at the expense of a decreased ability to explore freely; three participants mentioned they would prefer Sandbox for *“trying out random things and see if the model can cope”* (P3), and *“I feel more at liberty to play with language outside the the Chain”* (P6). They suggested they would prefer a combination of both systems: *“when there’s more ambiguity I prefer the sandbox to explore first, but once I have a clear goal, I would use the Chaining to steer myself towards a fixed number of function blocks.”* (P13)

Inspired by these concerns, we envision future research to focus on relaxing certain structural constraints and providing guidance on LLM Chain creation and refinement, which we detail later in Discussion (Section 6.7).

6.6 Case Studies

Beyond the user study tasks, LLM Chaining has the potential to enable a wide range of complex applications. We illustrate how Chaining could support more diverse applications through two case studies in the domains of software development and accessibility, using the same model in our user study.

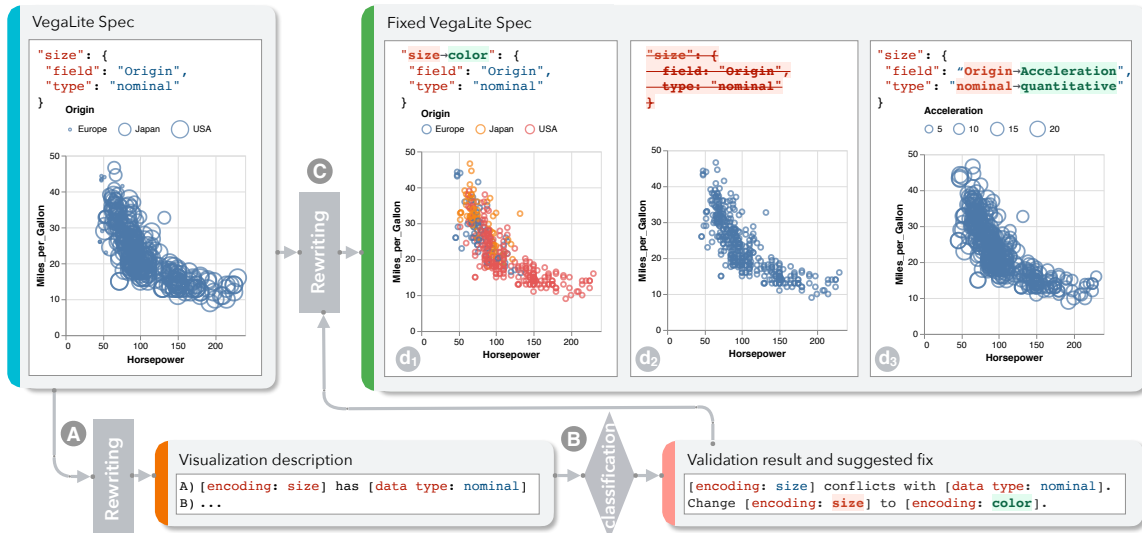


Figure 6.7: An example for Chaining-based VegaLite bug fixing (simplified; the full Chain is in Appendix C.3). (A) We first **rewrite** the ● JSON format specs into ● natural language descriptions to make it more parsable, then (B) **classify** the descriptions to ● validate design constraints and suggest fixes, and (C) finally **rewrite** the ● final spec based on the suggested fix. While the LLM generates the fix in d_1 , users may also choose to produce d_2 and d_3 , both of which can fix the ● validated issue just as effectively.

6.6.1 Case 1: Visualization code debugging

In this case study on visualization code debugging, we uncover how intermediate data points in a Chain can become useful, especially when the end goal of the task is unclear. Unlike typical code syntax errors, when a visualization violates design constraints [Moritz et al., 2018], there are usually multiple valid solutions that cannot be objectively ranked. For example, the ● original visualization (using VegaLite specifications [Satyanarayan et al., 2016]) in Figure 6.7 has a single violation, *i.e.*, circle size is continuous and thus should not be used to represent the discrete (nominal) field “Origin.” However, there may be multiple ways to resolve the issue [Chen et al., 2021b], such as using color instead of size (d_1), removing size information altogether (d_2), or changing the data encoded to a continuous “Acceleration” field (d_3). Thus, LLMs should reason about the violated constraints for users to adjust the fixes. However, in a single run of an LLM, this reasoning can be challenging,

as LLMs have trouble parsing visualization specs in JSON formats (see LLM Challenge C.3 in Section 6.3.1).

We thus created a Chain (see Figure 6.7) that (A) *rewrites* the JSON format in natural language, (B) *classifies* and validates the descriptions, and (C) *rewrites* the spec. To explore how the Chain performs in practice, we took examples from VizLinter [Chen et al., 2021b], used five pairs of erroneous and fixed specs as few-shot prompt examples, and tested the Chain on another five cases. One author with sufficient visualization knowledge determined that the Chain correctly revealed the violated constraints for all the test cases, and provided useful fixes for two of them. We also tried running a single pass of the LLM for comparison on the same examples, using multiple prompt designs. We observed that output from the single-passes tended to be consistently worse, with at most one correct reasoning. This is possibly due to parsing difficulty (see LLM Challenge C.3), as well as the inability to disentangle the sub-tasks of validation and rewriting (C.1). In contrast, each Chain step was highly scoped, increasing the chance that the intermediate data would be correct.

6.6.2 Case 2: Assisted Text Entry

We further demonstrate how Chaining could enable the branching logic in *assisted text entry*. This is based on a real industry use case that aims to speed up gaze input by requiring fewer character inputs [Majaranta and R  ih  , 2007, Rough et al., 2014, Adhikary et al., 2021]. Ideally, a user (*e.g.*, person using Alternative and Augmentative Communication technology) would express a full sentence through short abbreviations that an LLM would automatically expand. However, there are too many possible expansions to disambiguate, *e.g.*, “LTSGCHKITOT” could mean “Let’s go check it out,” “Let’s get coffee and have a chat,” “Let’s get some chicken in the old town,” etc. Thus, the end user often needs to resolve the ambiguity or adjust the input.

With Chaining, we enable interactive disambiguation through gradual expansion and if-else logic. As shown in Figure 6.8, if the user input is a shorthand (*e.g.*, “LTSG”), the LLM should expand it to possible matching phrases (“Let’s go”, “Let’s get”), which the user can select from. However, if the input is already a phrase, the LLM should instead auto-complete it (“Let’s go” may trigger “check it out.”) If the desired option does not appear, the user can also insert additional short-hands

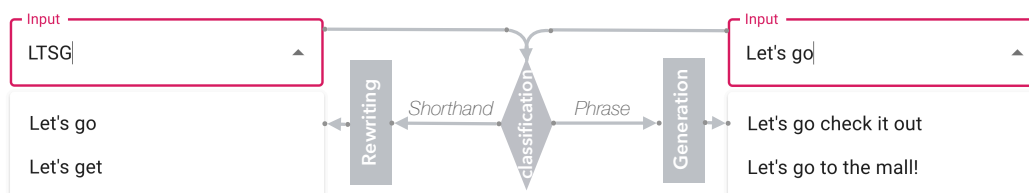


Figure 6.8: An example of Chaining-based assisted text entry (the full Chain is in Appendix C.3). To produce better full sentences, we **classify** the input text to switch between expanding shorthands (through **Rewrite**) and auto-completing phrases (through **Generation**). By wrapping the complex Chaining logic in a simple text field, we provide intuitive interactions for end users.

for the model to expand again, *e.g.*, “Let’s go CHKITOT”, which would exclude expansions starting with “Let’s get.” The switch between shorthand expansion and auto-completion enables better prediction on the full text, which would be nontrivial for a single prompt, given the different natures of the two branches. This case also provides a glimpse into how LLM Chains can help prototype applications with complex logic but simple interactions (elaborated in the next section).

6.7 Discussion & Future Directions

Our work is a first step towards improving human-LLM interaction through Chaining. We found that it not only raises the ceiling of what LLMs can meaningfully support, but also boosts transparency, controllability and debuggability — key concerns when interacting with generative AI [Amershi et al., 2019, Bommasani et al., 2021]. Interestingly, we achieved this purely by reshaping the interaction mechanism, without any need to retrain the model. This suggests that LLMs to date may already have the potential to support human-AI collaborations on many complex tasks, if their latent potential can be better realized through thoughtful interaction design. Below, we discuss the implications of our studies, as well as future research directions.

Chaining as a new paradigm of control on multiple model units. Contrary to recent work in human-AI interaction, which primarily examined how to increase AI controllability through exposing knobs within a model [Louie et al., 2020, Narayan et al., 2019], our work opens up

the possibility of steering AI using *the model itself* as units to control. In other words, beyond controlling properties within a single model unit, users may be able to achieve new kinds of control through manipulating how *multiple* model runs interact with one another, including: how modifications to upstream model units cascade, how to isolate changes between model units, and how to improve user inputs by comparing the effectiveness of parallel model runs. As language models grow in size and capability, they may ironically allow users to treat them as smaller entities of abstraction — serving as building blocks towards larger human goals.

We envision the HCI community innovating more types of building blocks that a model can provide, as well as the ways they can be combined. In particular, model units could be used not only to accomplish sub-tasks, but also to more thoroughly aid in the task decomposition design and debugging process. To overcome users' own systematic omissions [Wu et al., 2021], an upstream unit could be designed to help users create sub-tasks to begin with, similar to metaprompting [Reynolds and McDonell, 2021]. Or, model units could serve as checkpoints along the Chain to ensure data correctness (similar to assertions in code). Moreover, while the Chains in this chapter consisted of only LLM steps, alternative designs may also interleave LLM steps with human-computation steps, depending on which roles each collaborator could best fill.

Chaining for rapid prototyping of integrated applications Chaining also opens up new possibilities for designing AI-infused applications. With LLMs' easy adaptation to natural language prompts, users could conceivably already prototype custom ML functionality with lower effort, as they bypass the otherwise necessary but expensive process of collecting data and designing models upfront [Bommasani et al., 2021]. Chaining further accelerates this design process. Taking advantage of interactions between multiple LLM steps, developers could build multiple Chains to envision possible flows of how an application may be used, and then perform A/B testing on those Chains. For example, in the case of assisted text entry (Section 6.6.2), developers could quickly prototype what might happen if end users were allowed to provide more context: *e.g.*, if the user is “having a meeting in 5 minutes,” then “Let’s go” is more likely than “Let’s get” for the abbreviation “LTSG.” They could test this interaction by adding an additional layer of input to the shorthand

expansion step.

One might argue that, because each run of an LLM involves some computational overhead, chaining may introduce additional costs that need to be weighed against their benefits. However, as indicated above, a key benefit of chaining is that it could flexibly power a wide range of prototypes and applications, *without* the need to train or build bespoke, single-purpose AIs. Thus, we believe the saved efforts outweigh the cost.

Balancing between structured scaffolding and free exploration While Chaining provided guardrails and scaffolding for helping users accomplish the task at hand, it also limited their ability to explore freely. Yet, experimenting, tinkering, and interacting are key to users forming mental models for AI [Narayan et al., 2019]. One way to balance between structure and exploration is to loosen structural constraints *within* steps. For example, it may be useful to permit users to customize prompts within each step in a Sandbox-like environment, and to define their own input and output parsers. In other words, rather than providing a full implementation of steps, a Chain could define the API with input-output types, and ask users to fill in the implementations for each step. Or, a small Sandbox could be provided along-side the Chaining interface, for users to occasionally use when they need to experiment with a new approach.

Meanwhile, though our studies mostly explored how humans *use* pre-defined LLM Chains, a natural follow-up question becomes whether end users can effectively *author* their own LLM Chains. Indeed, one potential downside of Chaining is that it may *decrease* transparency if the pre-built Chain does not match the way a user would naturally break down the task (mentioned in Section 6.5.3). We believe our operations can serve as a starting point for future work on authoring. With the templates, users could instantiate an LLM step by defining the data layers and selecting the operations. In our study, most participants were able to spot deficiencies and refine the default Chains accordingly. Thus, we envision that a set of generic default Chains could help onboard end users to the idea of LLM Chaining, and inspire them to author more tailored Chains. We leave end user authoring of Chains to future work.

Enhancing LLM Chain design and refinement Our work centered mostly on moderately complex tasks that can be naturally broken down. However, decomposition might be less straightforward in some cases [Kim et al., 2017]. Tasks with more complex interdependence may lose coherence and quality if they are split into independent sub-parts. For example, in the Review task (Figure 6.1), we treated the different problems independently. However, if the problems are interrelated, keeping them together would promote more effective suggestions (*e.g.*, not engaging *and* speaks too quietly). Moreover, while users had the option of excluding specific data layers along the way (*e.g.*, the original review in Figure 6.1 is not fed into the final step), the information loss may also lead to task distortion or compression [Reynolds and McDonell, 2021]. In light of these issues, future work could investigate how to assist users in crafting the steps of a Chain to maximize its utility [Kittur et al., 2011]. For example, users could be provided strategic guidance on iterative Chain improvements, such as using paired comparisons and version control of Chain edits to help users decide whether to keep or further decompose an existing step.

6.8 Conclusion

In this work, we introduce the notion of “Chaining” multiple LLM steps together, such that the output of one step is the input to the next. We present an interactive system where users can modify these Chains, along with their intermediate results, in a modular way. We find that Chaining not only enhanced the quality of the task outcome, but also improved user satisfaction, with an increased sense of control and collaboration, a greater perception of transparency of the LLM system, and more support of the user’s thought processes. Furthermore, we envision with case studies that LLM Chaining may be advantageous for complex AI-infusion applications and in cases where intermediate reasoning is more important than the final output. We encourage future work to explore how LLMs can serve other kinds of building blocks, how Chains can be used in rapid prototyping, and strategies that can help users build and iterate on Chains.

Chapter 7

CONCLUSION AND FUTURE WORK

This thesis presents frameworks, interactive systems, and NLP models that allow humans to interactively debug and correct NLP models. On the one hand, we help AI experts run scalable and testable analyses on models in development so they can diagnose and address model weaknesses before models are released. This offline analysis acts as a guardrail, making it less likely for end users to encounter unwanted model behavior. In this region, we first designed Errudite for supporting error analysis; then, we further extended the two building blocks — systematic grouping and counterfactual rewriting — to various other model development stages by building various interactive tools. These tools are tailored to mitigate human biases from different perspectives, with Tempura exposing inherent dataset properties, Polyjuice increasing the diversity of counterfactuals, and Scattershot supporting iterative task refinement. On the other hand, we also help end users collaborate with deployed AIs in a transparent and controllable manner so they can detect and overwrite AI errors in real-time. As an example, we created AI Chaining to improve interaction between end users and large language models. Throughout the thesis, we show that it is promising to support human-centered AI development and usage, with a data-centered approach.

Extending beyond NLP. Though we mostly focus on text data, the methods have the potential to be generalized. For example, computer vision models share similar issues of being overly stable or overly sensitive to small perturbations, and similar approaches of counterfactual rewriting could potentially reveal otherwise unnoticed spurious features. Existing generators already allow for creating natural-looking images or audios [Li et al., 2020, Bau et al., 2018, Yao et al., 2021], suggesting the potential of creating pairs of minimally different examples in other modalities. However, unlike in the text domain, where edits are discrete even in model-readable embedding spaces, it is unclear how manipulations should be controlled on a super-pixel level. Future work

should look into counterfactual rewriting on the continuous data space. For example, instead of only creating a minimally perturbed image, we can imagine synthesizing a reference adversarial image with similar synthesized noise [Goodfellow et al., 2014], so the adversarial image and the edited image can form a pair with controlled differences.

7.1 Discussion: The Importance of Data

As mentioned above, we take a first step in emphasizing data in model design, evaluation, and usage. We believe data has more potential to bridge between humans and AI, and envision that our ability to make models more context-sensitive and reflective of real-life goals will be enhanced if we can better explore and exploit the rich context and connections between data points. Below, we briefly list some potential directions for enriching datasets.

Explicate data complexity and associations. Hierarchies exist within model capabilities. For chatbots to master debate skills, they must first learn how to gather factual information. Based on such observation it seems possible to build datasets with explicit connections between simple and difficult cases (*e.g.*, decomposing a multi-hop question into a series of single-step sub-questions [Yang et al., 2018]). With the data itself reflecting the task complexity, we can potentially train and test models to gradually master simple-to-complex capabilities.

Track the evolution of the scene with data. Real-world use cases are rarely stationary. Users adjust their behavior as they interact with AIs (*e.g.*, using only short, clear commands to instruct virtual assistants). These changes usually can be reflected through *e.g.*, distribution shift [Koh et al., 2021, Zhong et al.]. Through longitudinal studies, we might be able to study the long-term impact of the deployed model on society, and augment existing datasets with temporal information. Then, by adjusting the data weights and altering the model iteration goals based on changes in user beliefs and actions, we can potentially ensure that future model updates are compatible with users' previous experiences [Bansal et al., 2019b].

Identify and outsource subjective data. While most models are trained to be standard for everyone, people frequently have subjective interpretations on the task, the use case, or on even ethical positions. To make the model more broadly applicable, we encourage future work to look

into separating *objective* knowledge and *subjective* opinions. Specifically, we should prioritize embedding *e.g.*, syntactic structure, knowledge graph, etc. into large pre-trained models, but make the model either retrieval-based [Nakano et al., 2021] or allow it to go through lightweight prompt tuning [Lester et al., 2021] per user. Hopefully by investigating what type of information is worth outsourcing, we can better tie models to their use scenarios.

Of course, data fixing is not the solution for all. With large language models pre-trained on millions of texts, rigorously analyzing the pre-training data distribution becomes impractical, and any data augmentation becomes trivial at this scale. While lightweight tuning seems promising for model improvement on downstream tasks, debugging such models might require more sophisticated model architectures or training methods, *e.g.*, reweighing existing data or changing the objective function, etc. We discuss more about distinguishing data vs. model error/fixing below.

7.2 Opportunities for Future Research

With the foundation built in this thesis, we also identify a number of dimensions that can affect the model analysis and debugging process. We discuss three aspects below: who is debugging the model, how they are debugging it, and what metric they should use.

7.2.1 Debugging in-the-wild.

In contrast to our prior work, which assumes a clear distinction between experts running offline analysis as they develop models, and end users collaborating in real time with deployed models, in reality the human roles are much more blurred: End users also spontaneously conduct model behavioral testing (*i.e.* submitting test cases to the model API to see how it behaves) when confronted with unexpected errors [Shen et al., 2021]. Sometimes, they even come together organically to collectively test model behaviors.

Two interesting research questions arise from their proactive debugging: First, how can we help end users exploit the patterns they are trying to test? In light of the emergence of large models like GPT-3 [Brown et al., 2020] or DALL-E 2 [Radford et al., 2021] as valuable ideation sources, it

seems promising to build tools that can support end users in identifying and creating data points that reflect their own expectations on models. We encourage future research to design and evaluate more sophisticated grouping and perturbation 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. Second, how can we aggregate information collected in the wild? Although some work (including AI Chains, Chapter 6) allows users to self-assess and correct the model outputs in use, they rarely offer support on returning the error back to developers. Consequently, the status-quo of end-user feedback remains user-filed tickets [Madaio et al., 2022] or complaints on social media [Shen et al., 2021], which are hardly efficient and can harm public relations. It would be beneficial to develop platforms that automatically collect user interactions and mine erroneous model behaviors so as to embed end users more deeply into the model iteration process, and connect the last step in Figure 1.1 back to the first. One challenge may be ranking crowdsourced errors by severity; Since end users may be more likely to make fun of models than fix them seriously, identifying real use cases based on user profiles, for instance, may prove helpful.

7.2.2 *Instructive and actionable debugging.*

The ultimate goal of debugging is to *correct* model errors after discovering them. Our work emphasizes more on identifying problems, and relies on augmented data for fixing them. However, as we have discussed in Chapter 4, while counterfactual data augmentation brings consistent gains across tasks, the improvements on some datasets are not as significant. In fact, plain augmentation can sometimes introduce additional bias to the model, improving one aspect at the expense of another [Ribeiro and Lundberg, 2022]. Future work should look into enhancing the value of counterfactuals for training robust models, by adding explicit terms in the loss function that compare counterfactuals with original data [Teney et al., 2020], or by implementing other forms of contrastive learning.

Other than data augmentation, it is also possible to fix models through partial retraining [Mitchell et al., 2021] or hyperparameter tuning [Khodak et al., 2021]. Addressing identified errors with the right fix becomes crucial: How can we tell if a specific set of errors is a result of a data

issue or a model issue? What are the steps to determine whether a single erroneous pattern exists (and an instructional patch is sufficient [Murty et al., 2022]) or whether multiple input patterns are lacking (leading to data augmentation)? To achieve the best model development outcomes, researchers should design an integrated development environment (IDE) that offers targeted warnings and assertions, guesses of the root causes, potential solutions, as well as integrated workflows for testing whether the fixes applied were indeed effective. One potential first step could be to quantify the impact of different model fixing behaviors, by surveying the results of ablation studies in the modeling literature [Meyes et al., 2019], as well as metrics that quantify dataset difficulty [Swayamdipta et al., 2020] or bias [Gardner et al., 2021]. The whole model development loop will be smoother if we can *teach* developers to link errors to most effective fixing actions with a cheatsheet summarizing this knowledge.

7.2.3 *Measuring human-AI performance.*

In Chapter 6, we hinted at two interesting “error” types: (1) *Subjective errors*, where the model is not strictly wrong, but it still does not meet users’ subjective requirements (*e.g.*, the rewritten paragraph in Figure 6.1, though mediocre, did move in the right direction); (ii) *Partial errors*, *i.e.*, the model makes “almost-there” suggestions that the user feels comfortable accepting and editing [Vaithilingam et al., 2022]. As these error types suggest, in human-generator interaction scenarios, correctness is rarely black and white. Instead, the evaluation should be more user-oriented and also usage-procedure-oriented. In other words, instead of trying to calculate certain typical metrics, it may be worth exploring human interaction metrics. Recently, CoAuthor [Lee et al., 2022], a dataset in HCI for benchmarking human-generator co-writing, has started to include data about user interaction and clickstreams. If we observe and analyze clickstreams, we might be able to determine people’s strategies as they “debug” based on their own needs and mental models.

In addition, the “correctness” of a model is also affected by the user’s intentions. As users revise their writing with a generator-based assistant, fluency and factuality become paramount; whereas creativity plays a much greater role when users ideate with the model. Breaking down overarching accuracies or BLEU scores into concrete evaluation metrics that are aligned with human objectives

is an important next step, which would ensure the human objective and the evaluation metric is important maximizing AIs' strength in the desired dimension.

BIBLIOGRAPHY

A recipe for arbitrary text style transfer with large language models. <https://www.gwern.net/GPT-3>. Accessed: 2021-08-01.

Jiban Adhikary, Jamie Berger, and Keith Vertanen. Accelerating text communication via abbreviated sentence input. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6574–6588, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.514. URL <https://aclanthology.org/2021.acl-long.514>.

Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the behavior of visual question answering models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1955–1960, Austin, Texas, 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1203. URL <https://aclanthology.org/D16-1203>.

Mehdi Allahyari, Seyedamin Pouriye, Mehdi Assefi, Saied Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. *ArXiv preprint*, abs/1707.02919, 2017. URL <https://arxiv.org/abs/1707.02919>.

Satya Almasian, Dennis Aumiller, and Michael Gertz. Bert got a date: Introducing transformers to temporal tagging. *arXiv preprint arXiv:2109.14927*, 2021.

Saleema Amershi, Daniel S. Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi T. Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. Guidelines for human-ai interaction. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *Proceedings of the 2019 CHI Conference*

on *Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 3. ACM, 2019. doi: 10.1145/3290605.3300233. URL <https://doi.org/10.1145/3290605.3300233>.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: visual question answering. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2425–2433. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.279. URL <https://doi.org/10.1109/ICCV.2015.279>.

Jaime Arguello, Fernando Diaz, Jamie Callan, and Jean-Francois Crespo. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM, 2009.

Nabiha Asghar. Yelp dataset challenge: Review rating prediction. *ArXiv preprint*, abs/1605.05362, 2016. URL <https://arxiv.org/abs/1605.05362>.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *ArXiv preprint*, abs/2108.07732, 2021. URL <https://arxiv.org/abs/2108.07732>.

Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S Lasecki, Daniel S Weld, and Eric Horvitz. Beyond accuracy: The role of mental models in human-ai team performance. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 7, pages 2–11, 2019a.

Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2429–2437, 2019b.

Gagan Bansal, Tongshuang Wu, Joyce Zhou, Raymond Fok, Besmira Nushi, Ece Kamar, Marco Tulio Ribeiro, and Daniel Weld. Does the whole exceed its parts? the effect of ai

explanations on complementary team performance. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445717. URL <https://doi.org/10.1145/3411764.3445717>.

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B Tenenbaum, William T Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. *arXiv preprint arXiv:1811.10597*, 2018.

Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.463. URL <https://aclanthology.org/2020.acl-main.463>.

Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. Soy lent: a word processor with a crowd inside. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, pages 313–322, 2010.

Gregor Betz, Kyle Richardson, and Christian Voigt. Thinking aloud: Dynamic context generation improves zero-shot reasoning performance of gpt-2. *ArXiv preprint*, abs/2103.13033, 2021. URL <https://arxiv.org/abs/2103.13033>.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte

Khani, Omar Khattab, Pang Wei Kohd, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2021.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL <https://aclanthology.org/D15-1075>.

Gwern Branwen. Gpt-3 creative fiction. 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, Decem-*

ber 6-12, 2020, virtual, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html>.

Jay Budzik and Kristian J Hammond. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 5th international conference on intelligent user interfaces*, pages 44–51. ACM, 2000.

Daniel Buschek, Lukas Mecke, Florian Lehmann, and Hai Dang. Nine potential pitfalls when designing human-ai co-creative systems. *ArXiv preprint*, abs/2104.00358, 2021. URL <https://arxiv.org/abs/2104.00358>.

Carrie J Cai, Philip J Guo, James Glass, and Robert C Miller. Wait-learning: leveraging conversational dead time for second language education. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 2239–2244. 2014.

Carrie J. Cai, Shamsi T. Iqbal, and Jaime Teevan. Chain reactions: The impact of order on microtask chains. In Jofish Kaye, Allison Druin, Cliff Lampe, Dan Morris, and Juan Pablo Hourcade, editors, *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, pages 3143–3154. ACM, 2016. doi: 10.1145/2858036.2858237. URL <https://doi.org/10.1145/2858036.2858237>.

Carrie J. Cai, Emily Reif, Narayan Hegde, Jason D. Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda B. Viégas, Gregory S. Corrado, Martin C. Stumpe, and Michael Terry. Human-centered tools for coping with imperfect algorithms during medical decision-making. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 4. ACM, 2019. doi: 10.1145/3290605.3300234. URL <https://doi.org/10.1145/3290605.3300234>.

Julia Cambre, Scott R. Klemmer, and Chinmay Kulkarni. Juxtapeer: Comparative peer review yields higher quality feedback and promotes deeper reflection. In Regan L. Mandryk, Mark Hancock,

- Mark Perry, and Anna L. Cox, editors, *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, page 294. ACM, 2018. doi: 10.1145/3173574.3173868. URL <https://doi.org/10.1145/3173574.3173868>.
- Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Efficient diversification of web search results. *Proceedings of the VLDB Endowment*, 4(7):451–459, 2011.
- John M Carroll and Judith Reitman Olson. Mental models in human-computer interaction. *Handbook of human-computer interaction*, pages 45–65, 1988.
- Angel X Chang and Christopher D Manning. Sutime: A library for recognizing and normalizing time expressions. In *Lrec*, volume 3735, page 3740, 2012.
- Ernie Chang, Xiaoyu Shen, Hui-Syuan Yeh, and Vera Demberg. On training instance selection for few-shot neural text generation. *arXiv preprint arXiv:2107.03176*, 2021.
- Joseph Chee Chang, Saleema Amershi, and Ece Kamar. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In Gloria Mark, Susan R. Fussell, Cliff Lampe, m. c. schraefel, Juan Pablo Hourcade, Caroline Appert, and Daniel Wigdor, editors, *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*, pages 2334–2346. ACM, 2017. doi: 10.1145/3025453.3026044. URL <https://doi.org/10.1145/3025453.3026044>.
- Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the CNN/Daily Mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2358–2367, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1223. URL <https://aclanthology.org/P16-1223>.
- Derek Chen, Zhou Yu, and Samuel R Bowman. Learning with noisy labels by targeted relabeling. *arXiv preprint arXiv:2110.08355*, 2021a.

- Nan-Chen Chen and Been Kim. Qsanglyzer: Visual analytics for prismatic analysis of question answering system evaluations. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology*, 2017.
- Qing Chen, Fuling Sun, Xinyue Xu, Jiazhe Wang, and Nan Cao. Vizlinter: A linter and fixer framework for data visualization. *IEEE transactions on visualization and computer graphics*, 2021b.
- Vincent S. Chen, Sen Wu, Alexander J. Ratner, Jen Weng, and Christopher Ré. Slice-based learning: A programming model for residual learning in critical data slices. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9392–9402, 2019. URL <https://proceedings.neurips.cc/paper/2019/hash/351869bde8b9d6ad1e3090bd173f600d-Abstract.html>.
- Justin Cheng, Jaime Teevan, Shamsi T. Iqbal, and Michael S. Bernstein. Break it down: A comparison of macro- and microtasks. In Bo Begole, Jinwoo Kim, Kori Inkpen, and Woontack Woo, editors, *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*, pages 4061–4064. ACM, 2015. doi: 10.1145/2702123.2702146. URL <https://doi.org/10.1145/2702123.2702146>.
- Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: crowdsourcing taxonomy creation. In Wendy E. Mackay, Stephen A. Brewster, and Susanne Bødker, editors, *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI ’13, Paris, France, April 27 - May 2, 2013*, pages 1999–2008. ACM, 2013. doi: 10.1145/2470654.2466265. URL <https://doi.org/10.1145/2470654.2466265>.
- Lydia B Chilton, James A Landay, and Daniel S Weld. Humortools: A microtask workflow for writing news satire. *El Paso, Texas: ACM*, 2016.

Ke-Li Chiu and Rohan Alexander. Detecting hate speech with gpt-3. *ArXiv preprint*, abs/2103.12407, 2021. URL <https://arxiv.org/abs/2103.12407>.

Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002.

Jason Chuang, Margaret E. Roberts, Brandon M. Stewart, Rebecca Weiss, Dustin Tingley, Justin Grimmer, and Jeffrey Heer. TopicCheck: Interactive alignment for assessing topic model stability. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 175–184, Denver, Colorado, 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1018. URL <https://aclanthology.org/N15-1018>.

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

Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 845–855, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1078. URL <https://aclanthology.org/P18-1078>.

Elizabeth Clark, Anne Spencer Ross, Chenhao Tan, Yangfeng Ji, and Noah A Smith. Creative writing with a machine in the loop: Case studies on slogans and stories. In *23rd International Conference on Intelligent User Interfaces*, pages 329–340, 2018.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL <https://openreview.net/forum?id=H1edEyBKDS>.

Nicholas Davis, Chih-PIn Hsiao, Kunwar Yashraj Singh, Lisa Li, and Brian Magerko. Empirically studying participatory sense-making in abstract drawing with a co-creative cognitive agent. In

Proceedings of the 21st International Conference on Intelligent User Interfaces, pages 196–207, 2016.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>.

Chris Donahue, Mina Lee, and Percy Liang. Enabling language models to fill in the blanks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2492–2501, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.225. URL <https://aclanthology.org/2020.acl-main.225>.

Darren Edge, Elly Searle, Kevin Chiu, Jing Zhao, and James A. Landay. Micromandarin: mobile language learning in context. In Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare, editors, *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3169–3178. ACM, 2011. doi: 10.1145/1978942.1979413. URL <https://doi.org/10.1145/1978942.1979413>.

Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48, 2016.

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1608–1618, Sofia, Bulgaria, 2013. Association for Computational Linguistics. URL <https://aclanthology.org/P13-1158>.

Cristian Felix, Anshul Vikram Pandey, and Enrico Bertini. Texttile: an interactive visualization tool

for seamless exploratory analysis of structured data and unstructured text. *IEEE transactions on visualization and computer graphics*, 23(1):161–170, 2017.

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

Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

Adam Fourney and Susan T. Dumais. Automatic identification and contextual reformulation of implicit system-related queries. In Raffaele Perego, Fabrizio Sebastiani, Javed A. Aslam, Ian Ruthven, and Justin Zobel, editors, *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 761–764. ACM, 2016. doi: 10.1145/2911451.2914701. URL <https://doi.org/10.1145/2911451.2914701>.

Kuzman Ganchev, Keith Hall, Ryan McDonald, and Slav Petrov. Using search-logs to improve query tagging. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 238–242, Jeju Island, Korea, 2012. Association for Computational Linguistics. URL <https://aclanthology.org/P12-2047>.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2501. URL <https://aclanthology.org/W18-2501>.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh 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, Ally Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.117. URL <https://aclanthology.org/2020.findings-emnlp.117>.
- Matt Gardner, William Merrill, Jesse Dodge, Matthew E Peters, Alexis Ross, Sameer Singh, and Noah Smith. Competency problems: On finding and removing artifacts in language data. *arXiv preprint arXiv:2104.08646*, 2021.
- Sahaj Garg, Vincent Perot, Nicole Limtiaco, Ankur Taly, Ed H Chi, and Alex Beutel. Counterfactual fairness in text classification through robustness. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 219–226, 2019.
- Katy Ilonka Gero and Lydia B. Chilton. Metaphoria: An algorithmic companion for metaphor creation. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 296. ACM, 2019. doi: 10.1145/3290605.3300526. URL <https://doi.org/10.1145/3290605.3300526>.
- Mor Geva, Yoav Goldberg, and Jonathan Berant. Are we modeling the task or the annotator? an investigation of annotator bias in natural language understanding datasets. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1161–1166, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1107. URL <https://aclanthology.org/D19-1107>.
- Max Glockner, Vered Shwartz, and Yoav Goldberg. Breaking NLI systems with sentences that require simple lexical inferences. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 650–655, Melbourne, Australia,

2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2103. URL <https://aclanthology.org/P18-2103>.
- Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.
- Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- Spence Green, Jason Chuang, Jeffrey Heer, and Christopher D Manning. Predictive translation memory: A mixed-initiative system for human language translation. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 177–187, 2014.
- Sumit Gulwani and Prateek Jain. Programming by examples: Pl meets ml. In *Asian Symposium on Programming Languages and Systems*, pages 3–20. Springer, 2017.
- Sonal Gupta, Diana L MacLean, Jeffrey Heer, and Christopher D Manning. Induced lexico-syntactic patterns improve information extraction from online medical forums. *Journal of the American Medical Informatics Association*, 21(5):902–909, 2014.
- Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana, 2018a. Association for Computational Linguistics. doi: 10.18653/v1/N18-2017. URL <https://aclanthology.org/N18-2017>.
- Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New

- Orleans, Louisiana, 2018b. Association for Computational Linguistics. doi: 10.18653/v1/N18-2017. URL <https://aclanthology.org/N18-2017>.
- Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*, 15(1):55–86, 2007.
- Peter Hase and Mohit Bansal. Evaluating explainable AI: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5540–5552, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.491. URL <https://aclanthology.org/2020.acl-main.491>.
- Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 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)*, pages 473–483, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1044. URL <https://aclanthology.org/P17-1044>.
- Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*, 1992. URL <https://aclanthology.org/C92-2082>.
- Raphael Hoffmann, Luke Zettlemoyer, and Daniel S Weld. Extreme extraction: Only one hour per relation. *ArXiv preprint*, abs/1506.06418, 2015. URL <https://arxiv.org/abs/1506.06418>.
- Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. Understanding and visualizing data iteration in machine learning. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–13, 2020.
- Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miroslav Dudík, and Hanna M. Wallach. Improving fairness in machine learning systems: What do industry practitioners need? In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors,

- Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 600. ACM, 2019. doi: 10.1145/3290605.3300830. URL <https://doi.org/10.1145/3290605.3300830>.
- Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL <https://openreview.net/forum?id=rygQYrFvH>.
- Matthew K Hong, Adam Fourney, Derek DeBellis, and Saleema Amershi. Planning for natural language failures with the ai playbook. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2021.
- Enamul Hoque, Vidya Setlur, Melanie Tory, and Isaac Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE transactions on visualization and computer graphics*, 24(1):309–318, 2017.
- Mengdie Hu, Krist Wongsuphasawat, and John Stasko. Visualizing social media content with sententree. *IEEE transactions on visualization and computer graphics*, 23(1):621–630, 2017.
- Minghao Hu, Furu Wei, Yuxing Peng, Zhen Huang, Nan Yang, and Dongsheng Li. Read + verify: Machine reading comprehension with unanswerable questions. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 6529–6537. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33016529. URL <https://doi.org/10.1609/aaai.v33i01.33016529>.
- Cheng-Zhi Anna Huang, Hendrik Vincent Koops, Ed Newton-Rex, Monica Dinculescu, and Carrie J Cai. Ai song contest: Human-ai co-creation in songwriting. *ArXiv preprint*, abs/2010.05388, 2020a. URL <https://arxiv.org/abs/2010.05388>.

William Huang, Haokun Liu, and Samuel R. Bowman. Counterfactually-augmented SNLI training data does not yield better generalization than unaugmented data. In *Proceedings of the First Workshop on Insights from Negative Results in NLP*, pages 82–87, Online, 2020b. Association for Computational Linguistics. doi: 10.18653/v1/2020.insights-1.13. URL <https://aclanthology.org/2020.insights-1.13>.

Henrik Imberg, Johan Jonasson, and Marina Axelson-Fisk. Optimal sampling in unbiased active learning. In *International Conference on Artificial Intelligence and Statistics*, pages 559–569. PMLR, 2020.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. Adversarial example generation with syntactically controlled paraphrase networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana, 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1170. URL <https://aclanthology.org/N18-1170>.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark, 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1215. URL <https://aclanthology.org/D17-1215>.

Ellen Jiang, Edwin Toh, Alejandra Molina, Aaron Donsbach, Carrie J Cai, and Michael Terry. Geline and genform: Two tools for interacting with generative language models in a code editor. In *Adjunct Publication of the 34rd Annual ACM Symposium on User Interface Software and Technology*, 2021.

Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J. Cai. Prompt-based prototyping with large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

- Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL <https://aclanthology.org/P17-1147>.
- Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *ArXiv preprint*, abs/1602.02410, 2016. URL <https://arxiv.org/abs/1602.02410>.
- Kushal Kafle and Christopher Kanan. An analysis of visual question answering algorithms. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1983–1991. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.217. URL <https://doi.org/10.1109/ICCV.2017.217>.
- Daniel Kahneman and Amos Tversky. The simulation heuristic. Technical report, Stanford Univ CA Dept of Psychology, 1981.
- Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Polo Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1):88–97, 2018.
- Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. Wrangler: interactive visual specification of data transformation scripts. In Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare, editors, *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3363–3372. ACM, 2011. doi: 10.1145/1978942.1979444. URL <https://doi.org/10.1145/1978942.1979444>.
- Divyansh Kaushik, Eduard H. Hovy, and Zachary Chase Lipton. Learning the difference that makes A difference with counterfactually-augmented data. In *8th International Conference on Learning*

- Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL <https://openreview.net/forum?id=Sk1gs0NFvr>.
- Vahid Kazemi and Ali Elqursh. Show, ask, attend, and answer: A strong baseline for visual question answering. *ArXiv preprint*, abs/1704.03162, 2017. URL <https://arxiv.org/abs/1704.03162>.
- Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. CTRL - A Conditional Transformer Language Model for Controllable Generation. *ArXiv preprint*, abs/1909.05858, 2019. URL <https://arxiv.org/abs/1909.05858>.
- Fereshte Khani, Martin Rinard, and Percy Liang. Unanimous prediction for 100% precision with application to learning semantic mappings. *arXiv preprint arXiv:1606.06368*, 2016.
- Daniel Khashabi, Tushar Khot, and Ashish Sabharwal. More bang for your buck: Natural perturbation for robust question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 163–170, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.12. URL <https://aclanthology.org/2020.emnlp-main.12>.
- Mikhail Khodak, Renbo Tu, Tian Li, Liam Li, Maria-Florina F Balcan, Virginia Smith, and Ameet Talwalkar. Federated hyperparameter tuning: Challenges, baselines, and connections to weight-sharing. *Advances in Neural Information Processing Systems*, 34, 2021.
- Joy Kim, Sarah Serman, Allegra Argent Beal Cohen, and Michael S Bernstein. Mechanical novel: Crowdsourcing complex work through reflection and revision. In *Proceedings of the 2017 acm conference on computer supported cooperative work and social computing*, pages 233–245, 2017.
- Najoung Kim, Roma Patel, Adam Poliak, Patrick Xia, Alex Wang, Tom McCoy, Ian Tenney, Alexis Ross, Tal Linzen, Benjamin Van Durme, Samuel R. Bowman, and Ellie Pavlick. Probing what different NLP tasks teach machines about function word comprehension. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*, pages 235–249,

Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/S19-1026. URL <https://aclanthology.org/S19-1026>.

Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 43–52, 2011.

Janin Koch, Andrés Lucero, Lena Hegemann, and Antti Oulasvirta. May ai?: Design ideation with cooperative contextual bandits. In Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos, editors, *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, page 633. ACM, 2019. doi: 10.1145/3290605.3300863. URL <https://doi.org/10.1145/3290605.3300863>.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanus Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021.

Anand Pramod Kulkarni, Matthew Can, and Bjoern Hartmann. Turkomatic: automatic, recursive task and workflow design for mechanical turk. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

Souvik Kundu and Hwee Tou Ng. A question-focused multi-factor attention network for question answering. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 5828–5835. AAAI Press, 2018. URL <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17226>.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris

- Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019a. doi: 10.1162/tacl_a_00276. URL <https://aclanthology.org/Q19-1026>.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019b. doi: 10.1162/tacl_a_00276. URL <https://aclanthology.org/Q19-1026>.
- Tessa A. Lau, Steven A. Wolfman, Pedro M. Domingos, and Daniel S. Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53:111–156, 2003.
- Edith Law and Haoqi Zhang. Towards large-scale collaborative planning: Answering high-level search queries using human computation. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011. URL <http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3675>.
- Gyeongbok Lee, Sungdong Kim, and Seung-won Hwang. Qadiver: Interactive framework for diagnosing QA models. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 9861–9862. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33019861. URL <https://doi.org/10.1609/aaai.v33i01.33019861>.
- Mina Lee, Percy Liang, and Qian Yang. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. *arXiv preprint arXiv:2201.06796*, 2022.

- Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.
- VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.
- Ariel Levy, Monica Agrawal, Arvind Satyanarayan, and David Sontag. Assessing the impact of automated suggestions on decision making: Domain experts mediate model errors but take less initiative. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.
- David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer, 1994.
- Dianqi Li, Yizhe Zhang, Hao Peng, Liqun Chen, Chris Brockett, Ming-Ting Sun, and Bill Dolan. Contextualized perturbation for textual adversarial attack. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5053–5069, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.400. URL <https://aclanthology.org/2021.naacl-main.400>.
- Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5284–5294, 2020.
- Xiao Li. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1337–1345, Uppsala, Sweden, 2010. Association for Computational Linguistics. URL <https://aclanthology.org/P10-1136>.
- Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL <https://aclanthology.org/C02-1150>.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. Technical report, AI21 Labs, 2021.

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

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.165. URL <https://aclanthology.org/2020.findings-emnlp.165>.

Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. TurkIt: human computation algorithms on mechanical turk. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, pages 57–66, 2010.

Alisa Liu, Swabha Swayamdipta, Noah A Smith, and Yejin Choi. Wanli: Worker and ai collaboration for natural language inference dataset creation. *arXiv preprint arXiv:2201.05955*, 2022.

Haochen Liu, Joseph Thekinen, Sinem Mollaoglu, Da Tang, Ji Yang, Youlong Cheng, Hui Liu, and Jiliang Tang. Toward annotator group bias in crowdsourcing. *arXiv preprint arXiv:2110.08038*, 2021a.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *ArXiv preprint*, abs/2101.06804, 2021b. URL <https://arxiv.org/abs/2101.06804>.

Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for machine reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1694–1704, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1157. URL <https://aclanthology.org/P18-1157>.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv preprint*, abs/1907.11692, 2019. URL <https://arxiv.org/abs/1907.11692>.

Shayne Longpre, Yu Wang, and Chris DuBois. How effective is task-agnostic data augmentation for pretrained transformers? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4401–4411, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.394. URL <https://aclanthology.org/2020.findings-emnlp.394>.

Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J. Cai. Novice-ai music co-creation via ai-steering tools for deep generative models. In Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik, editors, *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pages 1–13. ACM, 2020. doi: 10.1145/3313831.3376739. URL <https://doi.org/10.1145/3313831.3376739>.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *ArXiv preprint*, abs/2104.08786, 2021. URL <https://arxiv.org/abs/2104.08786>.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting*

- of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, 2011. Association for Computational Linguistics. URL <https://aclanthology.org/P11-1015>.
- Aman Madaan, Amrith Setlur, Tanmay Parekh, Barnabas Poczos, Graham Neubig, Yiming Yang, Ruslan Salakhutdinov, Alan W Black, and Shrimai Prabhunoye. Politeness transfer: A tag and generate approach. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1869–1881, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.169. URL <https://aclanthology.org/2020.acl-main.169>.
- Nishtha Madaan, Inkit Padhi, Naveen Panwar, and Diptikalyan Saha. Generate your counterfactuals: Towards controlled counterfactual generation for text. *Proceedings of the AAI Conference on Artificial Intelligence*, 2021.
- Michael Madaio, Lisa Egede, Hariharan Subramonyam, Jennifer Wortman Vaughan, and Hanna Wallach. Assessing the fairness of ai systems: Ai practitioners’ processes, challenges, and needs for support. *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW1):1–26, 2022.
- Päivi Majaranta and Kari-Jouko Räihä. Text entry by gaze: Utilizing eye-tracking. *Text entry systems: Mobility, accessibility, universality*, pages 175–187, 2007.
- Eric Malmi, Aliaksei Severyn, and Sascha Rothe. Unsupervised text style transfer with padded masked language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8671–8680, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.699. URL <https://aclanthology.org/2020.emnlp-main.699>.
- Andrew McCallum. Information extraction: Distilling structured data from unstructured text. *Queue*, 3(9):4, 2005.

- Tom McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1334. URL <https://aclanthology.org/P19-1334>.
- Prem Melville and Raymond J Mooney. Diverse ensembles for active learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 74, 2004.
- Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks. *arXiv preprint arXiv:1901.08644*, 2019.
- Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1 – 38, 2019. ISSN 0004-3702. doi: <https://doi.org/10.1016/j.artint.2018.07.007>. URL <http://www.sciencedirect.com/science/article/pii/S0004370218305988>.
- Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. Question answering through transfer learning from large fine-grained supervision data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 510–517, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-2081. URL <https://aclanthology.org/P17-2081>.
- Sewon Min, Victor Zhong, Richard Socher, and Caiming Xiong. Efficient and robust question answering from minimal context over documents. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1725–1735, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1160. URL <https://aclanthology.org/P18-1160>.
- Sewon Min, Xinxu Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

- Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. *ArXiv preprint*, abs/2104.08773, 2021. URL <https://arxiv.org/abs/2104.08773>.
- Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.
- Alaa Mohasseb, Mohamed Bader-El-Den, and Mihaela Cocea. Analysis of the syntactical structure of web queries. In *2018 International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 2, pages 557–562. IEEE, 2018.
- Dominik Moritz, Chenglong Wang, Greg L Nelson, Halden Lin, Adam M Smith, Bill Howe, and Jeffrey Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics*, 25(1): 438–448, 2018.
- John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 119–126, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.16. URL <https://aclanthology.org/2020.emnlp-demos.16>.
- Pramod Kaushik Mudrakarta, Ankur Taly, Mukund Sundararajan, and Kedar Dhamdhere. Did the model understand the question? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1896–1906, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1176. URL <https://aclanthology.org/P18-1176>.
- Shikhar Murty, Christopher D Manning, Scott Lundberg, and Marco Tulio Ribeiro. Fixing model bugs with natural language patches. In *ACL Workshop on Learning with Natural Language Supervision*, 2022.

- Aakanksha Naik, Abhilasha Ravichander, Norman Sadeh, Carolyn Rose, and Graham Neubig. Stress test evaluation for natural language inference. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2340–2353, Santa Fe, New Mexico, USA, 2018. Association for Computational Linguistics. URL <https://aclanthology.org/C18-1198>.
- Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- Preslav Nakov, Sara Rosenthal, Zornitsa Kozareva, Veselin Stoyanov, Alan Ritter, and Theresa Wilson. SemEval-2013 task 2: Sentiment analysis in Twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA, 2013. Association for Computational Linguistics. URL <https://aclanthology.org/S13-2052>.
- Hegde Narayan, Jason D Hipp, Yun Liu, Michael Emmert-Buck, Emily Reif, Daniel Smilkov, Michael Terry, Carrie J Cai, Mahul B Amin, Craig H Mermel, et al. Similar image search for histopathology: Smily. *NPJ Digital Medicine*, 2(1), 2019.
- Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.
- Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1018. URL <https://aclanthology.org/D19-1018>.
- Jakob Nielsen. *Usability engineering*. Elsevier, 1994.
- Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021.

Besmira Nushi, Ece Kamar, Eric Horvitz, and Donald Kossmann. On human intellect and machine failures: Troubleshooting integrative machine learning systems. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1017–1025. AAAI Press, 2017. URL <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/15032>.

Joe O’Connor and Jacob Andreas. What context features can transformer language models use? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 851–864, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.70. URL <https://aclanthology.org/2021.acl-long.70>.

Changhoon Oh, Jungwoo Song, Jinhan Choi, Seonghyeon Kim, Sungwoo Lee, and Bongwon Suh. I lead, you help but only with enough details: Understanding user experience of co-creation with artificial intelligence. In Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox, editors, *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, page 649. ACM, 2018. doi: 10.1145/3173574.3174223. URL <https://doi.org/10.1145/3173574.3174223>.

Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Gpt3-to-plan: Extracting plans from text using gpt-3. *ArXiv preprint*, abs/2106.07131, 2021. URL <https://arxiv.org/abs/2106.07131>.

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

OpenAI. Prompt design 101. <https://beta.openai.com/docs/introduction/prompt-design-101>, 2021. Accessed: 2021-08-07.

Marius Pasca and Benjamin Van Durme. What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI*, volume 7, pages 2832–2837, 2007.

Judea Pearl. Causal and counterfactual inference. *The Handbook of Rationality*, pages 1–41, 2018.

Remus Pop and Patric Fulop. Deep ensemble bayesian active learning: Addressing the mode collapse issue in monte carlo dropout via ensembles. *arXiv preprint arXiv:1811.03897*, 2018.

Ana-Maria Popescu and Oren Etzioni. Extracting product features and opinions from reviews. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 339–346, Vancouver, British Columbia, Canada, 2005. Association for Computational Linguistics. URL <https://aclanthology.org/H05-1043>.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4782–4793, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.432. URL <https://aclanthology.org/2020.acl-main.432>.

James Pustejovsky, Jessica Littman, Roser Saurí, and Marc Verhagen. Timebank 1.2 documentation. *Event London, no. April*, pages 6–11, 2006.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. Counterfactual story reasoning and generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5043–5053, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1509. URL <https://aclanthology.org/D19-1509>.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual

models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL <http://jmlr.org/papers/v21/20-074.html>.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016a. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016b. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>.

Marc’ Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL <http://arxiv.org/abs/1511.06732>.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. Data

programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishal Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL <https://aclanthology.org/D19-1410>.

Daniela Retelny, Michael S Bernstein, and Melissa A Valentine. No workflow can ever be enough: How crowdsourcing workflows constrain complex work. *Proceedings of the ACM on Human-Computer Interaction*, 1(CSCW):1–23, 2017.

Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

Marco Tulio Ribeiro and Scott Lundberg. Adaptive testing and debugging of NLP models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3253–3267, Dublin, Ireland, May 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.acl-long.230>.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco,*

CA, USA, August 13-17, 2016, pages 1135–1144. ACM, 2016. doi: 10.1145/2939672.2939778. URL <https://doi.org/10.1145/2939672.2939778>.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 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)*, pages 856–865, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1079. URL <https://aclanthology.org/P18-1079>.

Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. Are red roses red? evaluating consistency of question-answering models. In *Association for Computational Linguistics (ACL)*, 2019.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.442. URL <https://aclanthology.org/2020.acl-main.442>.

Marc-Antoine Rondeau and T. J. Hazen. Systematic error analysis of the Stanford question answering dataset. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 12–20, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2602. URL <https://aclanthology.org/W18-2602>.

Alexis Ross, Ana Marasović, and Matthew Peters. Explaining NLP models via minimal contrastive editing (MiCE). In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3840–3852, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.336. URL <https://aclanthology.org/2021.findings-acl.336>.

Alexis Ross, Tongshuang Wu, Hao Peng, Matthew Peters, and Matt Gardner. Tailor: Generating and perturbing text with semantic controls. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3194–3213, Dublin,

- Ireland, May 2022. Association for Computational Linguistics. URL <https://aclanthology.org/2022.acl-long.228>.
- Daniel Rough, Keith Vertanen, and Per Ola Kristensson. An evaluation of dasher with a high-performance language model as a gaze communication method. In *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, pages 169–176, 2014.
- Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*, 2021.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020. URL <https://aaai.org/ojs/index.php/AAAI/article/view/6399>.
- Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. “everyone wants to do the model work, not the data work”: Data cascades in high-stakes ai. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.
- Mattia Samory and Tanushree Mitra. Senpai: Supporting exploratory text analysis through semantic & syntactic pattern inspection. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 13, pages 452–462, 2019.
- Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.
- Roy Schwartz, Maarten Sap, Ioannis Konstas, Leila Zilles, Yejin Choi, and Noah A. Smith. The effect of different writing tasks on linguistic style: A case study of the ROC story cloze task.

- In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 15–25, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1004. URL <https://aclanthology.org/K17-1004>.
- Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.
- Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL <https://openreview.net/forum?id=HJ0UKP9ge>.
- Vidya Setlur, Melanie Tory, and Alex Djalali. Inferencing underspecified natural language utterances in visual analysis. In *IUI*, pages 40–51, 2019.
- Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.
- H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.
- Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Building bridges for web query classification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 131–138. ACM, 2006.
- Hong Shen, Alicia DeVos, Motahhare Eslami, and Kenneth Holstein. Everyday algorithm auditing: Understanding the power of everyday users in surfacing harmful algorithmic behaviors. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–29, 2021.
- Yelong Shen, Xiaodong Liu, Kevin Duh, and Jianfeng Gao. An empirical analysis of multiple-turn reasoning strategies in reading comprehension tasks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages

- 957–966, Taipei, Taiwan, 2017. Asian Federation of Natural Language Processing. URL <https://aclanthology.org/I17-1096>.
- Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. Auto-Prompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.346. URL <https://aclanthology.org/2020.emnlp-main.346>.
- Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pages 336–343. IEEE, 1996.
- Alison Smith-Renner, Ron Fan, Melissa Birchfield, Tongshuang Wu, Jordan L. Boyd-Graber, Daniel S. Weld, and Leah Findlater. No explainability without accountability: An empirical study of explanations and feedback in interactive ML. In Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguy, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik, editors, *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pages 1–13. ACM, 2020. doi: 10.1145/3313831.3376624. URL <https://doi.org/10.1145/3313831.3376624>.
- Adam Sobieszek and Tadeusz Price. Playing games with ais: The limits of gpt-3 and similar large language models. *Minds and Machines*, pages 1–24, 2022.
- Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, 2013. Association for Computational Linguistics. URL <https://aclanthology.org/D13-1170>.
- Ben Swanson, Kory Mathewson, Ben Pietrzak, Sherol Chen, and Monica Dinalescu. Story centaur:

- Large language model few shot learning as a creative writing tool. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 244–256, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-demos.29. URL <https://aclanthology.org/2021.eacl-demos.29>.
- Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. *arXiv preprint arXiv:2009.10795*, 2020.
- Jeniya Tabassum, Alan Ritter, and Wei Xu. Tweetime: A minimally supervised method for recognizing and normalizing time expressions in twitter. *arXiv preprint arXiv:1608.02904*, 2016.
- Alex Tamkin, Dat Nguyen, Salil Deshpande, Jesse Mu, and Noah Goodman. Active learning helps pretrained models learn the intended task. *arXiv preprint arXiv:2204.08491*, 2022.
- Bowen Tan, Zichao Yang, Maruan Al-Shedivat, Eric Xing, and Zhiting Hu. Progressive generation of long text with pretrained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4313–4324, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.341. URL <https://aclanthology.org/2021.naacl-main.341>.
- Jaime Teevan, Shamsi T Iqbal, Carrie J Cai, Jeffrey P Bigham, Michael S Bernstein, and Elizabeth M Gerber. Productivity decomposed: Getting big things done with little microtasks. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 3500–3507, 2016.
- Damien Teney, Ehsan Abbasnejad, and Anton van den Hengel. Learning what makes a difference from counterfactual examples and gradient supervision. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 580–599, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58607-2.

- Jonas Thiergart, Stefan Huber, and Thomas Übellacker. Understanding emails and drafting responses—an approach using gpt-3. *ArXiv preprint*, abs/2102.03062, 2021. URL <https://arxiv.org/abs/2102.03062>.
- Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *ArXiv preprint*, abs/2201.08239, 2022. URL <https://arxiv.org/abs/2201.08239>.
- Toan Tran, Thanh-Toan Do, Ian Reid, and Gustavo Carneiro. Bayesian generative active deep learning. In *International Conference on Machine Learning*, pages 6295–6304. PMLR, 2019.
- Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, 2022.
- Betty van Aken, Julian Risch, Ralf Krestel, and Alexander Löser. Challenges for toxic comment classification: An in-depth error analysis. In *Proceedings of the 2nd Workshop on Abusive Language Online (ALW2)*, pages 33–42, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5105. URL <https://aclanthology.org/W18-5105>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.
- Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.
- Gust Verbruggen, Vu Le, and Sumit Gulwani. Semantic programming by example with pre-trained models. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–25, 2021.

- Vasilis Verroios and Michael S Bernstein. Context trees: Crowdsourcing global understanding from local views. In *Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.
- Soumya Wadhwa, Khyathi Chandu, and Eric Nyberg. Comparative analysis of neural QA models on SQuAD. In *Proceedings of the Workshop on Machine Reading for Question Answering*, pages 89–97, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2610. URL <https://aclanthology.org/W18-2610>.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018a.
- Cunxiang Wang, Boyuan Zheng, Yuchen Niu, and Yue Zhang. Exploring generalization ability of pretrained language models on arithmetic and logical reasoning. *ArXiv preprint*, abs/2108.06743, 2021a. URL <https://arxiv.org/abs/2108.06743>.
- Peng Wang, Jiaming Xu, Bo Xu, Chenglin Liu, Heng Zhang, Fangyuan Wang, and Hongwei Hao. Semantic clustering and convolutional neural network for short text categorization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 352–357, Beijing, China, 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-2058. URL <https://aclanthology.org/P15-2058>.
- Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. Want to reduce labeling cost? gpt-3 can help. *arXiv preprint arXiv:2108.13487*, 2021b.
- Yizhong Wang, Kai Liu, Jing Liu, Wei He, Yajuan Lyu, Hua Wu, Sujian Li, and Haifeng Wang. Multi-passage machine reading comprehension with cross-passage answer verification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1918–1927, Melbourne, Australia, 2018b. Association for Computational Linguistics. doi: 10.18653/v1/P18-1178. URL <https://aclanthology.org/P18-1178>.

- Zhao Wang, Kai Shu, and Aron Culotta. Enhancing model robustness and fairness with causality: A regularization approach. *arXiv preprint arXiv:2110.00911*, 2021c.
- Matt Webb. GPT-3 is an idea machine. https://interconnected.org/home/2020/09/04/idea_machine, 2021. Accessed: 2021-08-23.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv preprint, abs/2201.11903*, 2022. URL <https://arxiv.org/abs/2201.11903>.
- Dirk Weissenborn, Georg Wiese, and Laura Seiffe. Making neural QA as simple as possible but not simpler. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 271–280, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1028. URL <https://aclanthology.org/K17-1028>.
- Sean Welleck, Ilya Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. Consistency of a recurrent language model with respect to incomplete decoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5553–5568, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.448. URL <https://aclanthology.org/2020.emnlp-main.448>.
- James Wexler. Facets: An open source visualization tool for machine learning training data. *Google Open Source Blog*, 2017.
- Sarah Wiegrefe and Yuval Pinter. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1002. URL <https://aclanthology.org/D19-1002>.
- John Wieting and Kevin Gimpel. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting*

of the Association for Computational Linguistics (Volume 1: Long Papers), pages 451–462, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1042. URL <https://aclanthology.org/P18-1042>.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL <https://aclanthology.org/2020.emnlp-demos.6>.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 747–763, Florence, Italy, 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1073. URL <https://aclanthology.org/P19-1073>.

Tongshuang Wu, Daniel S Weld, and Jeffrey Heer. Local decision pitfalls in interactive machine learning: An investigation into feature selection in sentiment analysis. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 26(4):1–27, 2019b.

Tongshuang Wu, Kanit Wongsuphasawat, Donghao Ren, Kayur Patel, and Chris DuBois. Tempura: Query analysis with structural templates. In Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguy, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik, editors, *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, pages 1–12. ACM, 2020. doi: 10.1145/3313831.3376451. URL <https://doi.org/10.1145/3313831.3376451>.

Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. Polyjuice: Generating

- counterfactuals for explaining, evaluating, and improving models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6707–6723, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.523. URL <https://aclanthology.org/2021.acl-long.523>.
- Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. *PromptChainer: Chaining Large Language Model Prompts through Visual Programming*. Association for Computing Machinery, New York, NY, USA, 2022a. ISBN 978145039156. URL <https://doi.org/10.1145/3491101.3519729>.
- Tongshuang Wu, Michael Terry, and Carrie J Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, CHI '21*, New York, NY, USA, 2022b. Association for Computing Machinery. ISBN 978145039157. doi: 10.1145/3491102.3517582. URL <https://doi.org/10.1145/3491102.3517582>.
- Yuxiang Wu, Matt Gardner, Pontus Stenetorp, and Pradeep Dasigi. Generating data to mitigate spurious correlations in natural language inference datasets. *arXiv preprint arXiv:2203.12942*, 2022c.
- Bingjun Xie, Jia Zhou, and Huilin Wang. How influential are mental models on interaction performance? exploring the gap between users' and designers' mental models through a new quantitative method. *Advances in Human-Computer Interaction*, 2017, 2017.
- Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.
- Runqi Yang, Jianhai Zhang, Xing Gao, Feng Ji, and Haiqing Chen. Simple and effective text matching with richer alignment features. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4699–4709, Florence, Italy, 2019. Association

- for Computational Linguistics. doi: 10.18653/v1/P19-1465. URL <https://aclanthology.org/P19-1465>.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.
- Xinwei Yao, Ohad Fried, Kayvon Fatahalian, and Maneesh Agrawala. Iterative text-based editing of talking-heads using neural retargeting. *ACM Transactions on Graphics (TOG)*, 40(3):1–14, 2021.
- Neal E Young. Greedy set-cover algorithms: 1974–1979; chvátal, johnson, lovász, stein. *Encyclopedia of algorithms*, pages 379–381, 2008.
- Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5564–5569, Florence, Italy, 2019a. Association for Computational Linguistics. doi: 10.18653/v1/P19-1559. URL <https://aclanthology.org/P19-1559>.
- Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018a.
- Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.
- Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, pages 627–648, 2020.
- Yan Zhang, Jonathon S. Hare, and Adam Prügel-Bennett. Learning to count objects in natural images for visual question answering. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018b. URL https://openreview.net/forum?id=B12Js_yRb.

- Yuan Zhang, Jason Baldridge, and Luheng He. PAWS: Paraphrase adversaries from word scrambling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1298–1308, Minneapolis, Minnesota, 2019b. Association for Computational Linguistics. doi: 10.18653/v1/N19-1131. URL <https://aclanthology.org/N19-1131>.
- Mengjie Zhao, Fei Mi, Yasheng Wang, Minglei Li, Xin Jiang, Qun Liu, and Hinrich Schütze. Lmturk: Few-shot learners as crowdsourcing workers. *arXiv preprint arXiv:2112.07522*, 2021.
- Ruiqi Zhong, Charlie Snell, Dan Klein, and Jacob Steinhardt. Describing differences between text distributions with natural language. *Red*, 5:99.
- Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Taxygen: A benchmarking platform for text generation models. In Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz, editors, *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 1097–1100. ACM, 2018. doi: 10.1145/3209978.3210080. URL <https://doi.org/10.1145/3209978.3210080>.

Appendix A

ADDITIONAL DETAILS FOR ERRUDITE

A.1 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/s_ODGuZU4G8.

A.1.1 VQA: Break down “How many”

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., 2018b] 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 A.1: 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 (Fig-

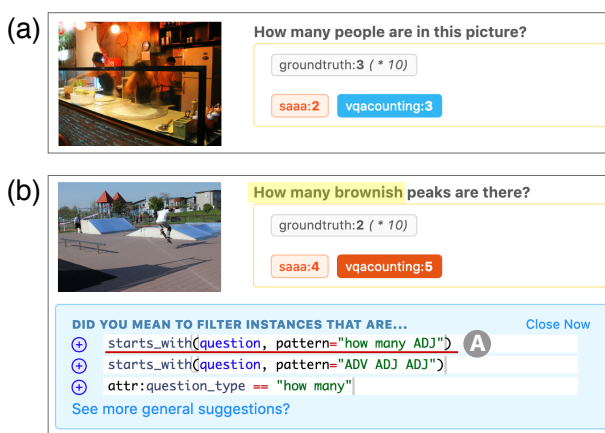


Figure A.1: Two “how many” examples: VQACounting improves on SAAA for instance (a), but predicts an even higher count in (b). Highlighting “how many brownish”, we create groups based on the suggested query A.

		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 A.2: VQACounting improves much more on how_many_NOUN, compared to how_many_ADJ, though many fewer instances follow the latter pattern.

ure A.1A) 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 A.2) 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, extracting the POS tags following "how many" into an attribute, we see "NOUN" drastically stands out, suggesting a very imbalanced data distribution (Figure A.3).

A.1.2 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

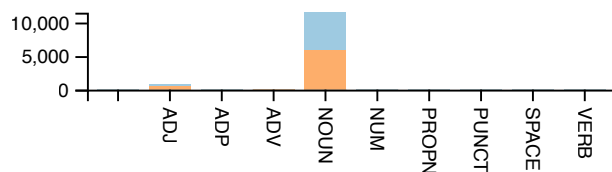


Figure A.3: Extracting the POS tag for the token immediately after "how many", we notice most instances follow a "how many NOUN" pattern.

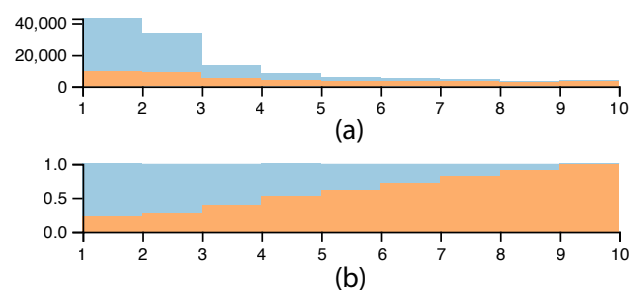


Figure A.4: The VQACounting model becomes worse as the count of distinct ground truth annotations (i.e., human disagreement) grows: (a) shows instance counts in each ground truth count bin; (b) with the counts normalized, emphasizing the incorrect proportions.



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 A.5: An instance with 8 distinct ground truth annotations, and negligible inter-annotator agreement.

humans. We count the unique ground truth annotations for each instance (`count(g)`), which results in the distribution shown in Figure A.4: 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 A.5, 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.1.3 MC: Incorrect Pre-processing

We used the following case as the tutorial demo in our user study (§2.5). When sorting instances by their *FI* score, instances like those in Figure A.6(a) appear. Due to

incorrect tokenization, BiDAF treats “1641-1679” as one token, and its mismatch with the ground truth “1679” evaluation (which is token-wise) will result in *FI* = 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 A.6(a), and also unexpected cases like Figure A.6(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=`

A	<p>Q: John Mayow died in what year? C: English chemist John Mayow (1641-1679) ...</p>
B	<p>Q: In what year did Good Company Player open? C: ...also opened nearby in 1987.[citation needed]</p>

Figure A.6: Two instances suspected to be wrong due to tokenization.

A	<pre>apply(exact_match(m), "add_space") == 1</pre> <p>Q: When did the Jin dynasty end? C: ...newly established Jin dynasty (1115-1234)(1115 - 1234)...</p>
B	<pre>apply(length(p(m)), "add_space") > 1</pre> <p>Q: When was John Gallagher born? C: ...John Gallagher (1919-1980)(1919 - 1980)...</p>
C	<pre>apply(exact_match(m), "add_space") == 0 and apply(length(p(m)), "add_space") == 1</pre> <p>Q: When did the Jin dynasty begin? C: ...newly established Jin dynasty (1919-1980)(1115 - 1234)...</p>

Figure A.7: Three types of changed predictions on instances generated with `add_space`.

"rule_name") runs the query function `func` on the new instances generated by rule "rule_name". We use the queries in Figure A.7 to further divide the 13 instances rewritten (the rule cannot edit additional cases like in Figure A.6(b)): 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.1.4 MC: "Why" Questions

We merge two cases from our user study in §2.5 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 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 *FI* 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.

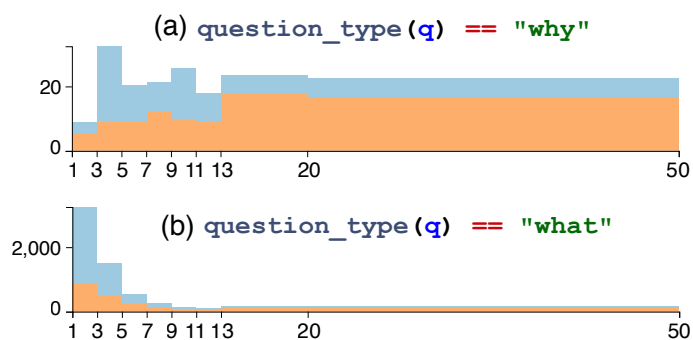


Figure A.8: The prediction length $\text{length}(p(m))$ is much longer for (a) only "why" questions than for (b) only "what" questions.

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

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

Figure A.9: A "why" question where the model ignored the apparent hint "because."

P2 found the example in Figure A.9 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 A.9 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.

A.2 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.

A.2.1 Query Ranking

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 Fig-

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 A.10: The illustrating example we used in the chapter; we repeat it here to explain our programming-by-demonstration heuristics. The scenario here assumes “John” is selected by a user.

ure A.10, and “How many brownish” is selected in Figure A.1. 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 A.10). 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 A.1, “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 A.1. A prediction (`p`) as in Figure A.10 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`):

<code>answer_type(g) == answer_type(p(m))</code>	1
<code>exact_match(m) == 0</code>	2
<code>is_correct_sent(m) == False</code>	3
<code>overlap(q, sentence(p(m))) ></code>	4
<code>overlap(q, sentence(g))</code>	5

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 sugges-

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 A.1: Patterns and their associated usefulness in Figure A.10 (top 3 lines) and Figure A.1 (bottom 2 lines)

tion, and rank their in-group error rate R_e and dataset coverage C_d , maximizing a usefulness score:

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

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 A.10 and Figure A.1, and their scores S_u , are shown in Table A.1.

A.2.2 Rewrite Rule Extraction

When a source x is edited to x' , we propose a set of rules $R = \{r_1, \dots, r_m\}$ 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.

Then, we apply every rule in the candidate set onto a random subset of instances $S = \{s_1, \dots, s_n\}, n = 100$. 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

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

DID YOU WANT TO GENERALIZE TO...	
who → what person	<input type="checkbox"/> keep
who created → what person created	<input type="checkbox"/> keep
who VERB → what person VERB	<input checked="" type="checkbox"/> keep
who created the → what person created the	<input type="checkbox"/> keep
who VERB DET → what person VERB DET	<input type="checkbox"/> keep

Figure A.11: Rewrite rules inferred from an edit on an individual instance.

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 A.11, an additional rule found in the candidate set from “Who” to “What person” was "NOUN" → "What person". By editing random NOUNs, 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.

A.3 Survey: Error Analysis Sample Sizes

Table A.2 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., 2017]	50
[Kundu and Ng, 2018]	50
[Hu et al., 2019]	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
<i>Average</i>	85.5

Table A.2: Surveyed papers and their error sample sizes.

A.4 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.

Appendix B

ADDITIONAL DETAILS FOR POLYJUICE

B.1 GPT-2 as Counterfactual Generator

B.1.1 Training Data and Parameters

We combine several datasets to finetune Polyjuice.

Contrast set. Authors of 10 existing NLP dataset each manually perturbed 100–1,000 instances to change the gold label, so to inspect a model’s local decision boundary [Gardner et al., 2020]. The perturbation patterns vary based on the tasks and the annotators, allowing us to learn diverse strategies. To make sure we can use the contrast set to evaluate the *Sentiment* model, we excluded the IMDb movie review from the training.

Counterfactually-augmented data (CAD). Kaushik et al. [2020] crowdsourced counterfactuals for IMDb movie review (1.7k), which we split into paired sentences to match the text length of other datasets. CAD’s perturbation patterns also vary based on the task, but can especially contribute to *negation*. As NLI is in our demonstrating applications, we did not use their 6.6k SNLI counterfactuals.¹

WinoGrande is a large-scale dataset of 44k instances for testing common sense problems [Sakaguchi et al., 2020]. It contains sentences that differ only by one trigger word (*e.g.*, one noun), making it most suitable for learning lexical exchanges.

ParaNMT-50M contains 50 million English-English sentential paraphrase pairs, covering various domains and styles of text, as well as different sentence structures [Wieting and Gimpel, 2018].

PAWS [Zhang et al., 2019b] contains pairs with high text overlaps, created through controlled

¹ Similarly, though *QQP* is suitable for training Polyjuice, we omitted it so *QQP* can be used in our evaluation.

Dataset	negation	quantifier	lexical	resemantic	insert	delete	restructure	shuffle	global
CAD	3,274	292	8,143	2,603	960	952	220	36	3,466
Contrast	336	436	1,607	1,291	589	586	275	149	877
HANS	50	0	0	0	3,926	3,926	494	1,602	2
ParaNMT	2,797	825	10,000	10000	6,442	6,205	5,136	1,417	10,000
PAWS	81	1,815	10,000	10000	3,630	3,403	4,551	10,000	10,000
WinoGrande	3,011	94	10,000	6,927	120	124	453	65	3184
<i>Crawled</i>	0	0	5,000	0	5,000	5,000	0	108	5,000
Total	9,549	3,462	44,750	30,821	20,667	20,167	11,129	13,377	32,529

Table B.1: The datasets used for finetuning Polyjuice, and the control code distributions.

word swapping, best demonstrating `shuffle` and `restructure`. We used its 49k Wikipedia parts.

HANS [McCoy et al., 2019], a challenge set for NLI, contains 10k pairs of premises and hypotheses created based on 10 heavily fallible syntactic templates, and therefore compensates rarer structural changes that may be missed by PAWS.

Crawled We additionally crawl naturally occurring sentence pairs from non-paired datasets boost some specific patterns and increase lexical diversity. This include (1) CommonGen [Lin et al., 2020], sentences with common sense concepts; (2) Natural Questions [Kwiatkowski et al., 2019b], collections of queries issued to Google Engines (and therefore involve various paraphrases of similar user intents), and (3) SQuAD [Rajpurkar et al., 2016a], whose paragraphs involve Wikipedia knowledge. We estimate *close* pairs using edit distance, and broadly accept those with less than 60% editing. To exclude tricky cases (*e.g.*, “how do I not be” can be incorrectly regarded as `negation` for “how do I recover it”), we only augment the most determined patterns: `lexical`, `insert`, `delete`, and `shuffle`.

To balance the distribution (Table B.1), for each dataset, we extract control codes from all the (x, \hat{x}) , and randomly sample up to 10,000 instances per codes. Still, `quantifier` and `negation` have less training data compared to other codes. Fortunately, these codes tend to be limited to more specific patterns (“more than”, “not”, “never”) when compared to “broad” codes like `lexical`, and thus even a small sample is enough to learn them. We finetuned an off-the-shelf GPT-2 model from Wolf et al. [2020] for 10 epochs with an initial learning rate $5e-5$, a batch size of 8, and a sequence length of 120 (but any LM can potentially be used). We select the best epoch based on the evaluation loss on a holdout set of size 5,000. The training took around 8 hours on two Titan RTXs.

B.1.2 Intrinsic Evaluation Details

Closeness and Diversity

Similar to Madaan et al. [2021], we compare the *diversity* and *closeness* of Polyjuice with alternative generators, *i.e.*, RoBERTa and T5, representing masked language models that prioritize word and span substitution, and original GPT-2, representing the standard generative model not conditioned on x . For a given x and its counterfactuals $\hat{\mathbf{X}}$, we approximate *diversity* using self-BLEU [Zhu et al., 2018] within $\hat{\mathbf{X}}$. Meanwhile, *closeness* is the average distance between x and every $\hat{x} \in \hat{\mathbf{X}}$, both with the normalized word level Levenshtein edit distance ([Levenshtein, 1966], used in MiCE [Ross et al., 2021]), and syntactic tree edit distance ([Zhang and Shasha, 1989] in GYC [Madaan et al., 2021]).

We run the three generators on 300 sentences in total. In GPT-2, we take the first two words of an x as the input context (prompt), limit the length of the generation to be similar to x , and collect 10 counterfactuals. As for RoBERTa and T5, we repeatedly perturb x for three times, each time randomly placing up to three [MASK] tokens, and ask the generator to generate 5 counterfactuals through beam search, following Ribeiro et al. [2020]. Polyjuice uses the same blank (mask) placement as in RoBERTa and T5, but we additionally enumerate through all control codes. For each x , we randomly sample 5 counterfactuals to form $\hat{\mathbf{X}}$ per generator.

As shown in Table 4.2, Polyjuice achieves a balance between diversity and closeness. Ideally, we would also like to compare Polyjuice with concurrent work [Madaan et al., 2021, Ross et al., 2021], but these are yet to be open-sourced and require extensive implementation or finetuning.

Controllability

To evaluate controllability, we compare Polyjuice with T5, and GPT-2 finetuned on prompts *without* codes (called Polyjuice -a), such that both baselines consider sufficient context. For each control code, we compare the *control success rate* of Polyjuice and Polyjuice-a on 300 prompts. For each prompt, we generate counterfactuals through beam search (beam = 5), and recompute the codes on the top three generated \hat{x} . We deem the control successful if at least one of the three recomputed

codes matches the desired control code (though in Polyjuice-a, we only measure whether the code naturally occurs in the uncontrolled generation.) The success rate increases by $26\% \pm 13\%$ across all control codes, ranging from `quantifier` (increasing 6%, from 50% to 56%) to `negation` (42%, from 5% to 47%). Non-finetuned T5 also achieves less control (success rate decreases by 33% on average.)

Common failure cases include (1) The control codes conflict with the blanks, *e.g.*, “a dog is embraced by a [BLANK]” would not respond to `negation`. (2) x does not have a corresponding pattern, *e.g.*, `shuffle` is not applicable to “the movie is good.” (3) certain salient patterns dominate the generation probability, *e.g.*, the model tends to perturb the quantifier “two” in “two dogs are running,” regardless of the code.

B.2 Additional Train & Eval Details, §4.3

B.2.1 MTurk Labeling Details

Procedure The study started with an introduction that explained the context and tasks. To familiarize crowdworkers with the task, we asked them to complete 1-2 training rounds, and explained the expected labels. Each annotator then completed 22 tasks, labeling 3 counterfactuals of a single example in each round, as in Figure B.1. The 22 rounds consisted of 20 actual labeling tasks and 2 extra “gold rounds” with known correct labels. The gold cases later served to filter low-quality crowdworkers. The median annotation time was around 15 minutes, and participants received \$2.5.

Participants. We recruited participants from MTurk, limiting the pool to subjects from within the US with a prior task approval rating of at least 97% and a minimum of 1,000 approved tasks.

Data quality. We applied two filtering strategies: (1) *High-quality worker.* We only kept data from participants whose median labeling time per round was more than 18 seconds and correctly labeled at least 4 gold counterfactuals (out of 6), or who correctly labeled all gold ones. (2) *Majority vote labeling.* We collected two annotations per counterfactual, and only kept those that at least one annotator deemed valid, and both annotators agreed on a particular class label. One of the authors labeled a subset of 100 \hat{x} on 100 x in *Sentiment*, and reached high agreement with the majority-voted

Reference Example

Old S1	Police officer with riot shield stands in front of crowd .
Old S2	A police officer stands in front of a crowd .
Label	Definitely True

Label the following! [Review the instructions!](#)

The **green color** highlights new words added in **New S2** , compared to **Old S2** in the **Reference example above**. • indicates something is deleted.

Old S1	Police officer with riot shield stands in front of crowd .
New S2	A police officer standing behind a crowd .
Valid?	<input type="radio"/> Invalid <input checked="" type="radio"/> Valid
Label	<input checked="" type="radio"/> Definitely False <input type="radio"/> May be True <input type="radio"/> Definitely True

Old S1	Police officer with riot shield stands in front of crowd .
New S2	A police officer stands next to a truck .
Valid?	<input type="radio"/> Invalid <input checked="" type="radio"/> Valid
Label	<input type="radio"/> Definitely False <input checked="" type="radio"/> May be True <input type="radio"/> Definitely True

Old S1	Police officer with riot shield stands in front of crowd .
New S2	A policeman stands in front of a crowd .
Valid?	<input type="radio"/> Invalid <input checked="" type="radio"/> Valid
Label	<input type="radio"/> Definitely False <input checked="" type="radio"/> May be True <input type="radio"/> Definitely True

Figure B.1: A sample labeling task: The crowdworkers annotate three counterfactuals based on their validity and class label, with respect to the original instance.

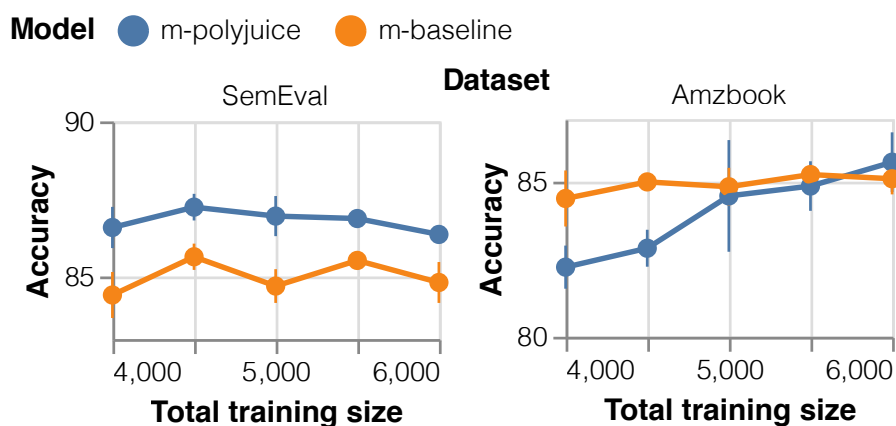


Figure B.2: The accuracy trend on two *Sentiment* datasets, as the total training dataset size ($m+n$) varies. The blue line shows an augmentation of $m = 2k$ counterfactuals, and the orange one represents the corresponding *m*-baseline. Though the counterfactuals remains useful on datasets like SemEval across all $m+n$, it appears too many counterfactuals may be harmful (Amzbook).

results ($\kappa = 0.77$, raw labeling agreement 88%).

B.2.2 Training Details & m/n Ratios, for §4.3.2

For each (m, n) , we created three samples of training data. Each sample was further averaged over four random seeds. For each run, we heuristically picked the initial learning rates $1e-5$, $2e-5$, $2e-5$ for *Sentiment*, *NLI* and *QQP*, and trained 20 epochs with a dropout rate of 0.1 and a batch size of 16. We selected the epoch that had the highest accuracy on the corresponding validation set, which takes 1/5 of the training data size, with the same ratio of m/n counterfactual and original examples.

We further explore ratios of added counterfactuals. Take *Sentiment* as an example: while the counterfactual remains effective on most datasets, it hurts the model performance on Amzbook when the counterfactual takes a large proportion (Figure B.2, Yelp followed a similar but more mild trend). We suspect that flipping out too much original data affects the data diversity, and in turn decreases the model performance. Similarly, Huang et al. [2020b] asserted that augmenting $n = 1.7k$ NLI data with $m = 6.6k$ counterfactuals did not improve model generalization accuracy.

REFERENCE EXAMPLE AND WORD IMPORTANCE

Old Q1 How can I help a friend experiencing serious depression ?

Old Q2 How do I help a friend who is in depression ?

Model predicts Duplicate (97.3% confident)

Model correct? Correct

ASK QUESTIONS!

You can edit Q2 and get model's prediction on variations of this example. Use it wisely, and try to come up with edits that will best help you understand the model behavior around the instance!

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I play with a friend who is in depression ?

Get prediction

QUERY RESULTS WILL BE DISPLAYED BELOW

The green color highlights new words added in New Q2, compared to Old Q2 in the Reference example above. • indicates something is deleted.

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I play with a friend who is in depression ?

Model predicts Non-duplicate (67.4% confident)

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help a friend who is in frustrated ?

Model predicts Non-duplicate (95.1% confident)

LABEL THE FOLLOWING!

The green color highlights new words added in New Q2, compared to Old Q2 in the Reference example above. • indicates something is deleted.

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I ask a friend who is in depression ?

Model will predict Non-duplicate Duplicate

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help a woman who is in depression ?

Model will predict Non-duplicate Duplicate

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help a friend who is in a relationship ?

Model will predict Non-duplicate Duplicate

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help someone who is depressed ?

Model will predict Non-duplicate Duplicate

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help a friend help me who is in depression ?

Model will predict Non-duplicate Duplicate

Old Q1 How can I help a friend experiencing serious depression ?

New Q2 How do I help a friend • ?

Model will predict Non-duplicate Duplicate

Figure B.3: A sample explanation task for §4.4

B.3 Additional Explanation Details §4.4

B.3.1 Selection Methods

Because SHAP weights reflect the average effect of masking a token t , we also focus on *word features that are abnormal on average*.

More concretely, we define the expected change-in-prediction for perturbing a token t to be the SHAP importance on it, $\mathbf{H}[D_f(t, x)] = s(t)$. In Figure 4.3, $s(t=\text{depression}) = 0.276$. The actual prediction change $D_f(t, x)$ is the weighted average of $|f_p(x) - f_p(\hat{x})|$ for all the \hat{x} that affect t ($\text{depression} \rightarrow \text{trouble}$, $\text{depression} \rightarrow \text{a mood}$), where $f_p(x)$ is the prediction probability of f on x . The weight corresponds to the number of words modified in \hat{x} : If $e(\hat{x})$ denotes the set of edited words in x , then $w(\hat{x}) = 1/|e(\hat{x})|$. Intuitively, the more words changed in \hat{x} , the less impact each word has; In Figure 4.3D, we regard “depression” to be responsible for half of the impact in $\text{in depression} \rightarrow \text{suicidal}$. We group \hat{x} based on their affected words $G_t = \{\hat{x} \mid t \in e(\hat{x})\}$. $D_f(t, x)$ then

becomes:

$$D_f(t, x) = \frac{1}{|G_t| + 1} \left(s(t) + \sum_{\hat{x} \in G_t} w(t) \cdot |f_p(x) - f_p(\hat{x})| \right)$$

The additional SHAP weight $s(t)$ acts as a smoothing factor to penalize outliers. Then the gap between the expectation and reality is:

$$\Delta D_f(t, x) = D_f(t, x) - \mathbf{H}[D_f(t, x)]$$

We first find the abnormal tokens: (1) t with small SHAP weight, but \hat{x} that change t experience large prediction change on average: $t_L = \arg \max_{t \in x} \Delta D_f(t, x)$, and (2) t with large SHAP weight, but \hat{x} with t changed usually have intact prediction: $t_U = \arg \max_{t \in x} -\Delta D_f(t, x)$.

Then, we use the most extreme cases within the groups of G_{t_L} and G_{t_U} as the concrete counterfactual explanations, based on their prediction change $|f_p(x) - f_p(\hat{x})|$, and the aggregated SHAP weights of all the changed tokens:

$$\hat{x}_L = \arg \max_{\hat{x} \in G_{t_L}} \left(|f_p(x) - f_p(\hat{x})| - \sum_{u \in r(\hat{x})} s(u) \right)$$

B.3.2 User Study Details

Figure B.3 shows the sample interface. Participants started by just seeing the reference example and the model query box on the left hand side. When they chose to start the task or after they had exhausted their ten query chances, the query box was disabled, the tasks on the right were displayed, and the participants completed the tasks. We compensated participants \$20 for the one hour study.

B.4 Additional Err. Analysis Details §4.5

B.4.1 Additional Case Study: Quantifiers

As a follow-up to Figure 4.6, we slice the data to find *entailment* instances that have numbers in the hypothesis sentence, and perturb their `quantifiers`. The extracted templates show that the model does not perform actual counting. When changing one number to another (`NUM` \rightarrow `NUM`), the model

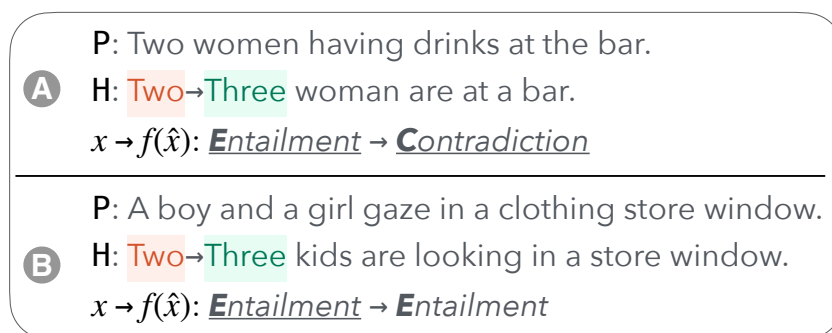


Figure B.4: The *NLI* model cannot perform the actual counting when the exact number is missing from *P*.

only flips the label in 64.7% cases, while we would expect all cases to be like in Figure B.4A. An inspection of instances indicates the model gets confused when the premise does not contain the same number explicitly. Indeed, when we filter for such instances (e.g. Figure B.4B), the label flip rate of **NUM** → **NUM** is lowered to 30.2%.

Further, the model only reacts to *some* quantifier phrase modifiers. **+at least** (“**at least** two women are at a bar”) will always still result in *entailment*, prediction, **+only** and **+exactly** flip the predicted label to *neutral* 90% of the time (“**exactly** two women are at a bar”), but the model only changes the prediction 52.6% of the time when we add **+more than** (“**more than** two women are at a bar”).

B.4.2 Representative Perturbation Templates

Similar to Wu et al. [2020], the process of finding representative perturbation patterns takes two steps:

Extract template. For each \hat{x} , we compare it with its x , and translate the perturbed spans into templates using different combinations of texts, lemmas, sparse and fine-grained part-of-speech tags. We optionally include surrounding contexts determined by the dependency tree structure (tokens that share the same parents as the perturbed span). For example, “is **not** reading” can result in templates t as fine-grained as **is reading** → **is not reading**, or as sparse as **+PART**. Meanwhile, “are **not** playing” also translates to **+PART** or **+not**, but not **is reading** → **is not reading**. As such, the \hat{x}

and templates form a many-to-many relationship: each \hat{x} generates multiple templates, and each template covers a different group of \hat{x} .

Select Representative Templates. To find representative changes, we prefer (1) templates that cover a large number of \hat{x} . Meanwhile, to avoid overfitting to one instance (*e.g.*, extracting a template `red` \rightarrow `ADJ` only because “red” is repeatedly perturbed in one x), we prefer (2) templates that perturb various unique x . We also prefer (3) finer-grained templates, to avoid being unnecessarily abstract (*e.g.*, to avoid abstracting “not” when it is the only PART changed.)

With these intuitions, we form the template selection as a weighted set coverage problem. We see the union of counterfactuals for each x , $\hat{\mathbf{X}}$, as the entire set of elements. Then, each template $t \in T = t_1, \dots, t_m$ represents a subset of $\hat{\mathbf{X}}$ that contains a number of counterfactuals $|t|$. We define the weight as $w(t) = g(t)/|t|_x$, where $|t|_x$ quantifies the unique original x covered by t , and $g(t)$ represents the sparsity of t (heuristically decreasing from `text` to `POS`). This way, templates that are too abstract or too focused on a certain x are penalized by having a high weight. We use a classic greedy algorithm [Vazirani, 2013] to select a subset of $T^* \subset T$, such that the aggregated coverage is maximized, and the weight is minimized.

Appendix C

ADDITIONAL DETAILS FOR AI CHAINS

C.1 Identifying LLM Primitive Operations

Primitive	Online demos
Info, extraction (9)	plan extraction [Olmo et al., 2021], arithmetic reasoning [Wang et al., 2021a], Keyword-extract, airport-code-extract, contact-info, color scale extractor, read code and answer questions, Summarize restaurant reviews (AI21), table question answering (AI21)
Classification (6)	hate speech detection [Chiu and Alexander, 2021], tweet-classifier, esrb rating, Automatically generating Request for Admissions, evaluate quiz answers, Classify news topics (AI21)
Rewrite (26)	program synthesis [Austin et al., 2021], Wordtune, generate database specific SQL code, parse-understructured-text, text to command, English to French, movie to emoji, tl;dr, sql-request, js-multi-line-to-one-line, js2python, html generation, description to app design description to todo list, Summarize-for-2nd-grade, Grammar-correction, third-person converter, rewrite as an attorney, Simplifying legal language, more polite, summarize famous people thoughts, speak in some personality, rewrite helper, mood color, De-jargonizer (AI21), Convert text to table (AI21)
Split points (1)	turn-by-turn directions
Composition (4)	Notes to summary review creator Description to ads Writing full emails from key points
Factual query (11)	add info to table, table computation, company to categories, factual answer, js-chatbot, ai-chatbot-tutor, sarcastic chatbot, imdb movie link, guess movie, Explain a word (AI21), Sports trivia (AI21)
Generation (8)	drafting email responses [Thiergart et al., 2021], Keyword 2 name, Generate poetry, spreadsheet generator, topic to horror story, Predict the outcome (AI21), project description generator (AI21), generate catchy headline (AI21),
Ideation (8)	scifi-booklist, essay outline, create study notes, interview questions, content creation for marketing, topic to quiz questions, VR fitness idea illustrator, blog post ideation (AI21)

Table C.1: A survey of 73 online demos that inspired the design of our operation, mostly from published manuscripts, the OpenAI official GPT-3 example page, the AI21 tutorial, and the demo collection repository. All the links are last accessed in 2021/08.

We reviewed 73 existing demos to identify promising LLM capabilities that may help overcome

the challenges above by scoping the inputs/outputs to be more amenable to what an LLM can handle.

First, we collected demos from LLM official websites (*e.g.*, GPT-3 and Jurassic), social media, and published case studies by searching for keywords including “GPT-3,” “language model,” “prompt,” etc. After removing some demos that were highly open-ended rather than targeted (*e.g.*, generic chatbots), we iteratively sorted the demos into eight LLM primitive operations, as shown in Table 6.1. For example, we distinguished between operations that had different expected data mappings (one-to-many *v.s.* many-to-one), and different application types (deterministic *v.s.* creative). We then grouped the primitives into three high level groups based on which LLM challenge they may help address. The groups also appear to be consistent with categories presented on the GPT-3 tutorial page,¹ which highlighted typical NLP tasks like *Classification*, *Generation* (*i.e.*, gather additional information in Table 6.1b), *Transformation* (*i.e.*, re-organization). Finally, we further refined the primitive categories and names based on feedback from three pilot users (one LLM expert and two UX engineers with basic knowledge of LLM prompting).

C.2 Additional Details for User Study

C.2.1 Questions in the Exit Survey

After completing the given task in both conditions, participants self-rated their experience on the following dimensions, in the form of seven-point Likert scale [Likert, 1932]. Each question was asked twice, once on *Sandbox* and once on *Chaining*. They described their reasoning along with the ratings.

- *Match goal*: I’m satisfied with my final results from [*Sandbox/Chaining*]; they met the task goal.
- *Think through*: The [*Sandbox/Chaining*] system helped me think through what kinds of outputs I would want to complete the task goal, and how to complete the task.

¹<https://beta.openai.com/docs/guides/completion/introduction>

- *Transparent*: The [Sandbox/Chaining] system is transparent about how it arrives at its final result; I could roughly track its progress.
- *Controllable*: I felt I had control creating with the [Sandbox/Chaining] system. I can steer the system towards the task goal.
- *Collaborative*: In [Sandbox/Chaining], I felt I was collaborating with the system to come up with the outputs.

Additionally, participants also answered the following two free form questions:

- *Difference*: What were the differences, if any, between the experience of completing the task using Sandbox and Chaining?
- *Vision*: If you were using language models in your work, in what situations would you prefer to use Sandbox? Chaining? Can you think of 1-3 concrete examples?

C.2.2 Clickstream Categorization

we log the text status before and after each round of model run. Through sequence match, we recover what's generated by the model after each run, and how the participants edit the text in between of two runs. We split the logs into: (1) RUN the model, (2) UNDO the model, where people removed the generations from the previous run, making the resulting text more similar to prior to the previous run, (3) FORMAT, where people only add or remove line split or formatting-related stopwords, (4) CREATE-CONTENT, where people only insert meaningful spans to the text, (5) CURATE-CONTENT, where people make all the other kinds of refinements on the existing text — in Chaining, this is a merge of changing the instruction, prefix, and the data entries. We also logged (6) CHANGE-TEMPERATURE to denote when people make non-text based change on the model input, *i.e.*, temperature.

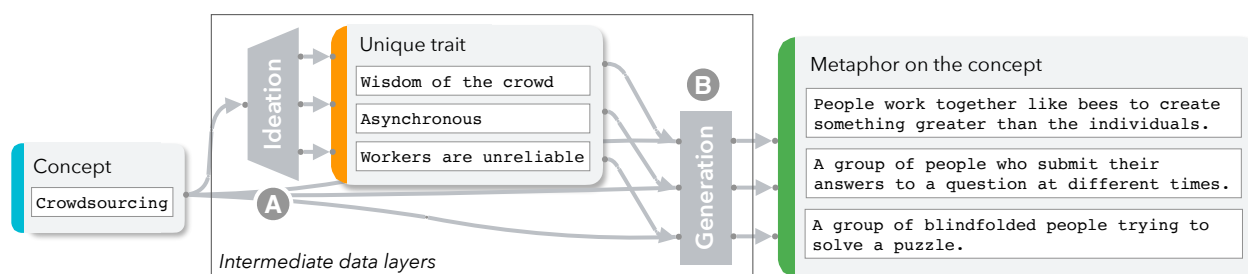


Figure C.1: The chain for acronym expansion. The steps include: (A) An **ideator** that brainstorms various **unique traits** for **the concept** (crowdsourcing). (B) For each trait, a **generator** creates a related **metaphor**.

On top of the logs, we define *consecutive runs* (in Figure 6.6A) as those in which users did not change anything after the previous run (or only add formatting through line changes or adding stopwords, *i.e.*, RUN+FORMAT). Otherwise, the logs are counted as humans making *edits*.

C.2.3 Case 0: Metaphor Creation (Used in tutorial)

Description. Create metaphors for the concept of *crowdsourcing*, so that we can explain the *different aspects* of crowdsourcing in a poetic way. The pipeline is as in Figure C.1.

A metaphor may look like:

In crowdsourcing, people are like bees; they work together to make honey.

With the *concept* being “crowdsourcing”, the *simile* being “bees”, and the *similar aspect* being “work together.”

Default baseline commands.

1. In the form of question answering,

Question: What is a good metaphor for crowdsourcing?

Answer: a swarm of bees.

2. In the form of command instruction,

Write a metaphor for the concept of crowdsourcing.

Concept: crowdsourcing

Metaphor: Crowdsourcing is like a game of chess. A crowdsourcer's skills, as in a chess player's skills, are combined with another person's skills to make something new.

3. List enumeration

The following is a list of metaphors on crowdsourcing.

1. Crowdsourcing is like a beehive - Many people (bees) contribute to a larger cause.

4. Few-shot example,

Concept: gratitude

Metaphor: gratitude is like a stream in that it's a force that can carry you along.

###

Concept: loss

Metaphor: loss is like a wing in that it's something you never wanted to lose, and it can take you away.

###

Concept: crowdsourcing

Metaphor: crowdsourcing is like a team sport in that it brings people to achieve one goal.

C.2.4 Case 1: Flashcard Creation

Description. Suppose you will be traveling to Paris next week, and you would like to create flashcards to learn about some basic French so you can have basic conversations with local people whenever you are in a non-English speaking region. Your goal is to create flashcards that are both **diverse** and **personalized** to your travel desires.

A flashcard may look like:

English: Where is a good restaurant?; French: Où est un bon restaurant?

Default baseline commands. we offer three versions:

1. In the form of question answering,

Question: What are some English and French sentence pairs useful for traveling to Paris?

Answers:

English: Where is a good restaurant?

French: Où est un bon restaurant?

2. In the form of list enumeration,

The following are English and French sentence pairs useful for traveling to Paris.

1. Que fais-tu ici? / What are you doing here?

3. Few-shot example,

English: I do not speak French.

French: Je ne parle pas français.

###

English: Where is a good restaurant?

French: Où est un bon restaurant?

###

English: Where's the bus station?

French: Où est la gare routière?

C.2.5 Case 2: Peer Review Paragraph Rewriting

Description. The following is a peer review paragraph on Alex's presentation. Please rewrite it to be more **friendly and constructive**. The paragraph should include **1-3 concrete and actionable suggestions** for **each** of Alex's presentation problems. Original feedback:

Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents.

An example output from the language model:

Alex, you have a lot of good ideas in your presentations. However, your slides are too cluttered with text. I would recommend that you use more graphics, pictures, and white space. Your slides should be simple enough that you could read your presentation off of the slides. This will allow you to engage the audience and not have to read from your notes. Also, you need to have a clear structure to your presentation. Lastly, you need to practice speaking. This will familiarize you with the material and reduce the likelihood of you meandering.

Default baseline commands. we offer two versions:

1. End-to-end version,

Given the feedback, rewrite it into a friendly paragraph with concrete suggestions on each of Alex's presentation problems.

Feedback: Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents.

Friendly paragraph: *[LLM generation]*

2. Two-step version, where we query LLM for improvement suggestions first, and then ask it to integrate the problem and the suggestion.

Alex could improve his presentation skills. He has too much text on his slides. His presentation meanders from topic to topic without a clear structure. He also does not engage with his audience when he presents.

Give Alex some suggestions on his presentation:

1. *[LLM generation]*

Write one friendly paragraph that covers all the presentation problems and suggestions: *[LLM generation]*

C.3 Full LLM Chains for Case Studies

These two figures are the detailed versions for the case studies. Figure C.2 is fore visualization bug fixing:

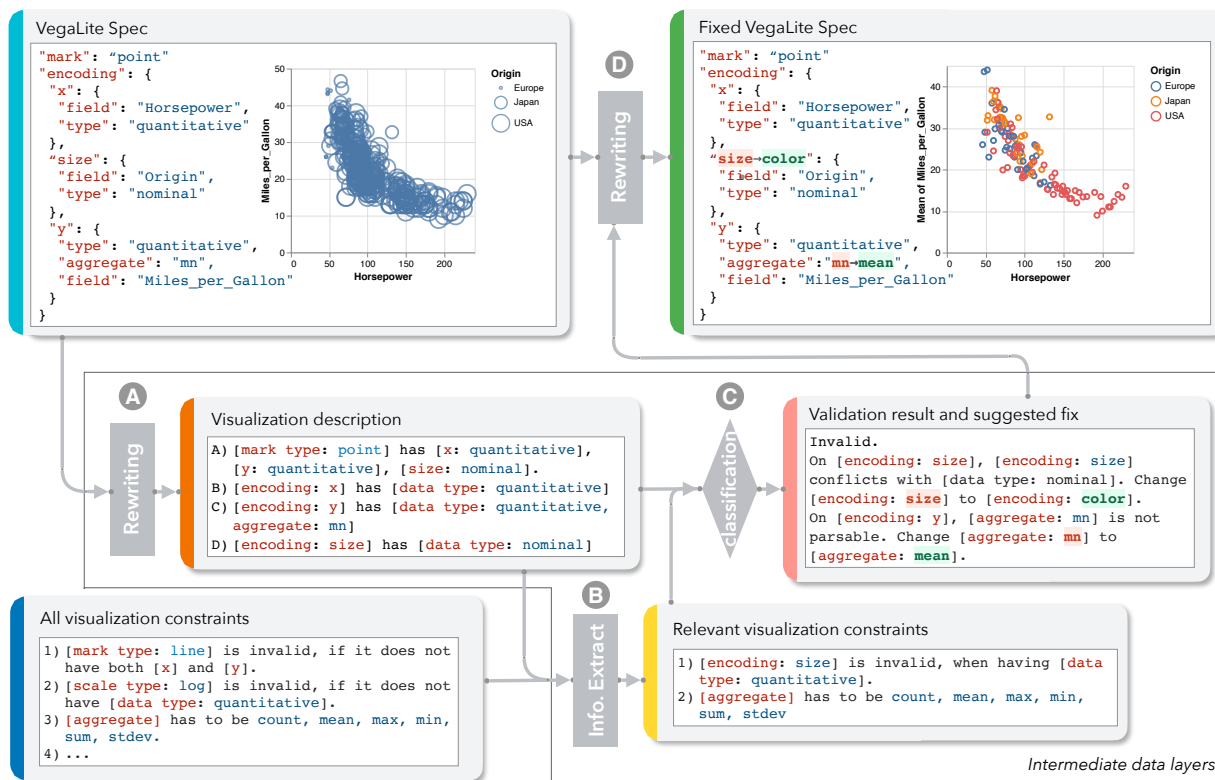


Figure C.2: The LLM Chain for visualization bug fixing (in VegaLite). The stages include: (A) A **Rewrite** step that transforms the ● *json format VegaLite spec* into ● *natural language description*, so to eliminate noise from data. (B) An **Information Extraction** step that locate ● *related visualization rules*. (C) A **Classification** step that verifies the description as either ● *valid or invalid* (with concrete errors and fixes). (D) A **Rewriting** step that generates ● *the fixed VegaLite spec* based on the ● *validity reasons*.

And Figure C.3 is for assisted text entry:

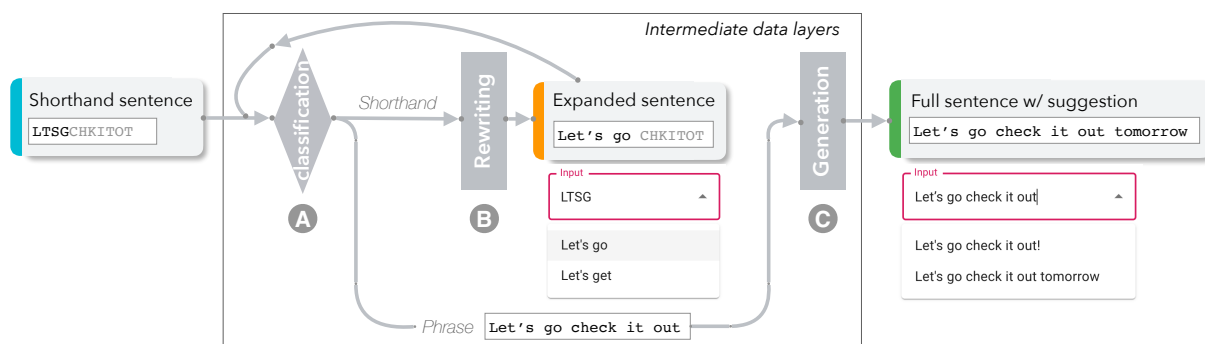


Figure C.3: The LLM Chain for assisted text entry. The stages include: (A) A **Classification** step that detects whether a ● *given sentence* ● *contains a shorthand or not*. (B) If there exists certain shorthand, a **Rewriting** step expands it, so we arrive at the ● *expanded sentence* which can become the context for additional shorthand inputs. For “LTSG”, it can be “Let’s go” or “Let’s get”, which relies on human selection. (C) Otherwise, a **Generation** step autocompletes ● *the sentence*.

C.4 The Full Implementation of Primitive Operations

Prompt template		Example	T
Classification: Assign the input based on limited categories. Most useful for branching logics and validation.			
Instruct	Classify if [detail-1] [detail-2].	Classify if the question is answerable .	0
Input	[prefix-1]: (str)	question: What is the square root of banana	
output	[prefix-2]: (str)	is answerable (Yes/No): <u>No</u>	
(a) Primitives for examining the given input , to judge its value and what to do next.			
Information Extraction: Gather some information from the context.			
Instruct	Given [detail-1], extract [detail-2].	Given text , extract airport codes for the cities .	0.2
Input	[prefix-1]: (string)	text: I want to fly from Los Angeles to Miami.	
output	[prefix-2]: (string)	airport codes: <u>LAX, MIA</u>	
Rewriting: 1-1 mapping that changes the input to more machine-readable formats (e.g. json to natural language).			
Instruct	Rewrite [detail-1] into [detail-2].	Rewrite the first-person text into third-person text .	0.3
Input	[prefix-1]: (string)	first-person text: I decide to make a movie	
output	[prefix-2]: (string)	third-person text: <u>He decides to make a movie.</u>	
Split Points: 1-N mapping that is particularly useful for splitting contexts.			
Instruct	Split [detail-1] into a list of [detail-2].	Split the descriptions on the direction into a list of turn-by-turn directions .	0.3
Input	[prefix-1]: (string)	Direction description: Go south on 95 until you hit Sunrise Blvd, then take it east to US-1 and head south.	
output	[prefix-2]: 1.(list of strings)	turn-by-turn directions: 1. <u>Drive south on 95.</u> 2. <u>Turn left onto Sunrise Blvd.</u> 3. <u>Turn left onto US-1 SE.</u>	
Compose Points: N-1 mapping, the reverse operation of decomposition; merge multiple results back.			
Instruct	Write one [detail-1] to cover all the [detail-2].	Write one review to cover all the restaurant name and notes .	0.5
Input	[prefix-1]: (list of strings)	Restaurant name: The Blue Wharf	
output	[prefix-2]: (string)	Short notes: 1. Lobster great; 2.noisy; 3.service polite Review: <u>The place is great if you like lobster. The noise level is a little high, but the service is polite.</u>	
(b) Primitives for reorganizing the input , and re-format it by parsing and expressing them in different ways.			
Factual Query: Ask the model for a fact.			
Instruct	Given [detail-1], find [detail-2].	Given the US state , find the population .	0.3
Input	[prefix-1]: (string)	US state: Washington	
output	[prefix-2]: (string)	Population: <u>7.6 million</u>	
Generation: Ask the model to do some creative “hallucination” on the input.			
Instruct	Given [detail-1], create [detail-2].	Given the topic , create a two-sentence horror story .	0.7
Input	[prefix-1]: (string)	topic: Breakfast	
output	[prefix-2]: (string)	two-sentence horror story: <u>He always stops crying when I pour the milk on his cereal. I just have to remember not to let him see his face on the carton.</u>	
Ideation: Ask the model for a list of ideas or examples.			
Instruct	Given [detail-1], the following is a list of [detail-2].	Given the interviewee , the following is a list of interview questions .	0.7
Input	[prefix-1]: (string)	Interviewee: A science fiction author	
output	[prefix-2]: 1.(list of strings)	Interview questions: 1. <u>What’s your favorite sci-fi book?</u> 2. <u>Who inspired you to start writing books?</u>	

(c) Primitives for **gathering additional clues from LLMs**.

Table C.2: We design a list of primitive building blocks, each with default prompting templates and temperatures, and group them by their intended objectives. Examples are taken from <https://beta.openai.com/examples>.