ScatterShot: Interactive In-context Example Curation for Text Transformation

Tongshuang Wu*
sherryw@cs.cmu.edu
Carnegie Mellon University
USA

Hua Shen
huashen218@psu.edu
Pennsylvania State University
USA

Daniel S. Weld
weld@cs.uw.edu
University of Washington &
Allen Institute for Artificial
Intelligence

Jeffrey Heer
jheer@cs.uw.edu
University of Washington
USA

Marco Tulio Ribeiro
marcotcr@microsoft.com
Microsoft Research
USA

ABSTRACT

The in-context learning capabilities of LLMs like GPT-3 allow annotators to customize an LLM to their specific tasks with a small number of examples. However, users tend to include only the most obvious patterns when crafting examples, resulting in underspecified in-context functions that fall short on unseen cases. Further, it is hard to know when "enough" examples have been included even for known patterns. In this work, we present ScatterShot, an interactive system for building high-quality demonstration sets for in-context learning. ScatterShot iteratively slices unlabeled data into task-specific patterns, samples informative inputs from underexplored or not-yet-saturated slices in an active learning manner, and helps users label more efficiently with the help of an LLM and the current example set. In simulation studies on two text perturbation scenarios, ScatterShot sampling improves the resulting few-shot functions by 4-5 percentage points over random sampling, with less variance as more examples are added. In a user study, ScatterShot greatly helps users in covering different patterns in the input space and labeling in-context examples more efficiently, resulting in better in-context learning and less user effort.

ACM Reference Format:
Tongshuang Wu, Hua Shen, Daniel S. Weld, Jeffrey Heer, and Marco Tulio Ribeiro. 2022. ScatterShot: Interactive In-context Example Curation for Text Transformation. In *IUI '23: Proceedings of the 28th Annual Conference on Intelligent User Interfaces.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In-context learning [70] is a property of Large Language Models (LLMs), where a user can "write" a transformation function via an (optional) short set of instructions and a few (input, output) examples. For example, writing a function that "translates" a holiday name (e.g. "Christmas") into its calendar date (e.g. "12/25") would previously require a complicated rule-based system capable of mapping various kinds of subtly different inputs (e.g. "Xmas", "Christmas day", etc) to a lookup table of dates. With LLMs like GPT-3 [7], the process is much simpler. A user can achieve the same functionality with a *prompt* (*i.e.,* a natural language instruction) that contains a small number (*e.g.,* two) of simple demonstrations, followed by a query (underlined): "Christmas => 12/25; Halloween => 10/31; Independence Day (US) =>". GPT-3 would take the prompt and return the right date "7/04" for this query. More impressively, LLM will also have some generalizability towards semantically relevant queries, *e.g.,* queries with abbreviations ("xmas => 12/25", "nye => 12/31"), misspellings ("s patrics day => 03/17"), lesser-known name variations ("All Saints' Eve => 10/31"), and holidays that might be overlooked (*e.g.,*"Harriet Tubman Day => 3/10"). The much reduced programming effort (compared to *e.g.,* rule-based systems) draws users' attention towards building their personalized in-context functions in various use scenarios, including code generation, question answering, creative writing, and others [36, 54, 64].

Although in-context learning has great potential, the quality of the learned function for non-trivial tasks depends on which in-context examples are used as demonstrations [32, 46]. Techniques for automatic example selection [30] depend on existing labeled datasets and tasks that can be evaluated automatically (*e.g.,* classification), and thus users "in the wild" rely on their own ingenuity and intuition when coming up with demonstrations [21]. Unfortunately, users tend to focus on the most obvious and memorable patterns for demonstration [18], leading to systematic omissions [66] and *underspecification* that might go unnoticed. As an example, in Figure 1 we use in-context learning to build a function to extract and normalize temporal information from a sentence [9]. Most users would provide demonstrations with common date formats (e.g. "Oct. 23, 1999"), and some might remember relative date references (e.g. "today"). However, some patterns are easy to miss, e.g. long-form dates with no capitalization or holidays (e.g. "nineteen ninety-six", "Thanksgiving Day" in Figure 1C), and the LLM may fail to learn them if they are omitted. Even sampling random examples from the unlabeled data might lead to the repetition of common patterns (Figure 1B) at the expense of demonstrating less-common ones. What

*The work was mostly done when the first authors was a PhD student at the University of Washington.

IUI '23, 2022, Sydney,Australia

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

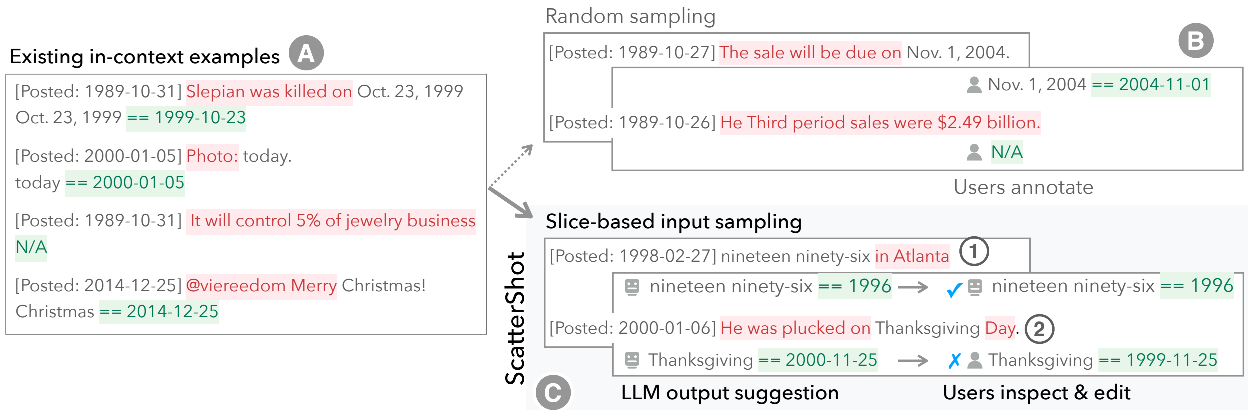

Figure 1: An overview of how human annotators can use ScatterShot to iteratively collect effective in-context examples for temporal expression extraction and normalization. The function extracts phrases with temporal meaning from sentences (*e.g.,*"Oct. 23, 1999" in "Slepian was killed on Oct. 23, 1999"), and normalizes them into standard formats ("Oct. 23, 1999 == 1999-10-23") — the red spans represent information deleted from the input, and the green ones represent information generated in the output. Given an in-context example set that is likely *underspecifying* the intended functionality (A), ScatterShot applies slice-based sampling to return unlabeled inputs that either have novel patterns or are difficult cases, and uses the existing examples to drive an LLM (*e.g.,* GPT-3) to suggest (possibly noisy) annotations, such that humans can correct the suggested annotations and possibly expand the in-context example bucket. Compared to random sampling and manual labeling (B), ScatterShot helps humans re-allocate annotation budgets towards informative examples, and increases the in-context function performance.

is worse, users may not know when they have provided enough examples, and whether there are any uncovered patterns in the unlabeled data. As a result, prior work summarized the two major pain points of prompting to be (1) the effort required to source examples for a prompt, and (2) the difficulty of evaluating whether a prompt is improving [22].

In this work, we present ScatterShot, an interactive system for building high-quality demonstration sets for in-context learning. In a nutshell, ScatterShot helps users find informative input examples in the unlabeled data, annotate them efficiently with the help of the current version of the learned in-context function, and estimate the quality of said function. In each iteration, ScatterShot automatically slices the unlabeled data into clusters based on task-specific *key phrases* [66, 69]. For example, given the existing examples in Figure 1A, it finds a cluster based on holiday key phrases ("Christmas", "Thanksgiving", etc.) and a cluster based on exact dates like "Oct. 23, 1999" (among others). ScatterShot keeps a running estimate of the error of each cluster, and thus prioritizes examples from clusters that have not yet been explored or learned effectively. It further uses the *stability* of the current in-context function with respect to minor changes in the prompt (e.g. ordering of in-context examples), prioritizing unlabeled examples that get different predictions with different prompt variations. Users are then presented with examples of underexplored clusters (*e.g.,* Figure 1 C_1), or hard examples of explored clusters (*e.g.,* C_2, hard because the past tense refers to the Thanksgiving date of the *previous* year). Instead of having to label demonstrations from scratch, users can either accept correct predictions from the current function (Fig 1 C_1) or make edits to fix wrong predictions (Fig 1 C_2). These additional labels are used to update the in-context function, such that the user explores the different possible input patterns in an interactive manner, without wasting resources on patterns that have already been learned.

We evaluate ScatterShot both in terms of sampling efficiency and support for human annotators. In simulation experiments, we compare the sampling strategy in ScatterShot to random sampling on two text transformation tasks contemplated in prior work: the data wrangling task illustrated in Figure 1 [9], and rewriting question-answer pairs into logically equivalent pairs in order to evaluate model consistency [44]. In both cases, we find ScatterShot improves performance on corresponding metrics (*e.g.,* Rouge-L, F1) by 4-5 percentage points, with less variance for various values of k demonstrations. Further, we conduct a within-subject user study in which 10 participants build in-context functions for the QA-pair rewriting task either (1) manually, (2) with the ScatterShot interface but random sampling, or (3) with the fully-featured ScatterShot. We show that ScatterShot's interface alone is an improvement, by offloading input selection and providing sample outputs. Moreover, the sampling strategy in the fully-featured ScatterShot helps users notice diverse input patterns, leading to improvements in the resulting in-context function. For example, participants who thought their in-context examples were sufficient when using random samples labeled *an additional* 1.4 times of examples after switching to full ScatterShot (as they found new patterns), which further improved the function test performance. We conclude the paper with insights into challenges and opportunities that arise from our experiments, including *e.g.,* explaining the sampling rationales, incorporating automated blind-spot detection, and the potential of using a ScatterShot setup to help users iteratively refine their task definition during data collection.

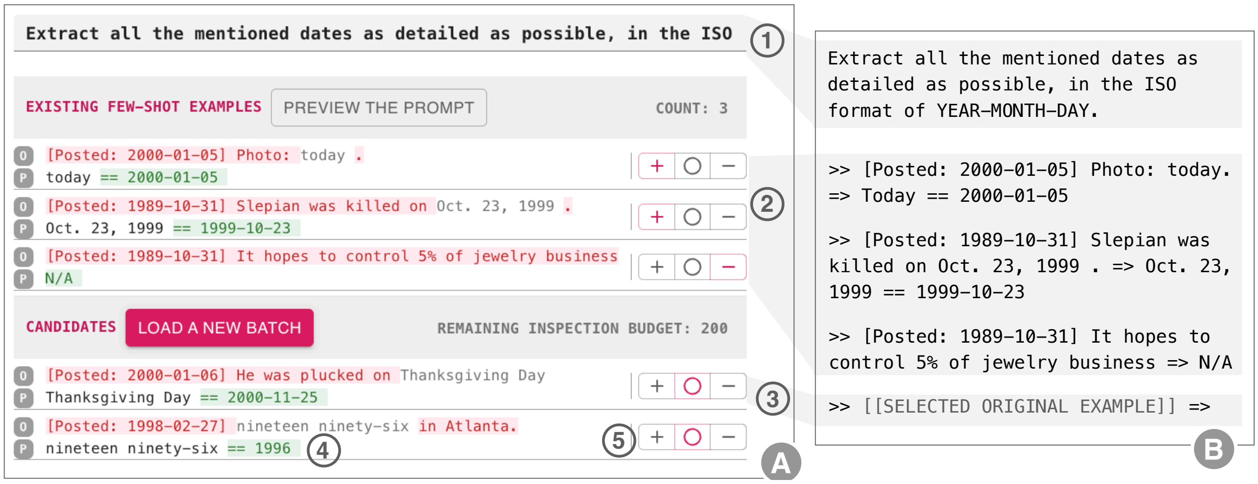

Figure 2: (A) The ScatterShot interface, with (A_1) task description, (A_2) existing in-context examples, and (A_3) candidate examples awaiting human inspection. Through interactions A_4 and A_5. Users can make edits to LLM outputs, sort the candidates into positive demonstrative examples (+), negative ones (-), or just not include the candidate (O). The description and the examples get transformed into raw text prompts. One set of in-context examples produces multiple prompts depending on how the examples are ordered; (B) shows a prompt with one possible ordering.

2 THE DESIGN OF SCATTERSHOT

The goal of ScatterShot is to help users iteratively find and label high-quality demonstrative examples to build effective in-context functions. In order to be effective, a function must be able to handle **common patterns** (*e.g.,* the temporal normalization function in Figure 1 must be able to handle common temporal expressions such as "today"), **without neglecting less common ones** (*e.g.,* holidays such as "Christmas"). Further, we want the process to be **cost-effective**, not wasting annotation effort on demonstrations that are redundant with already covered patterns. To achieve these goals, we design ScatterShot to respond to every user interaction by offering assistance in three areas:

- **Help the user discover previously unexplored patterns**. In each iteration, ScatterShot uses *existing* demonstrations and users' past interactions to cluster the remaining unlabeled data into task-specific slices. Such slices map the input space for users to explore.
- **Help the user prioritize the most informative examples**. ScatterShot uses the current in-context function to estimate the difficulty of slices *and* examples, prioritizing unexplored slices or slices and examples where the current function is not yet performing well. We call this variant of active learning *slice-based* sampling.
- **Minimize annotation cost**. Rather than providing a gold output label from scratch for each example, the user is presented with the best guess output from the *current* in-context function (updated at every step), which they either accept when correct or *edit* the incorrect parts.

We wrap these functionalities with a lightweight interface, where at each round, users are presented with a batch of unlabeled examples to be (potentially) added to the set of demonstrations. Thus, at each round, users get a "picture" of their current in-context function, and interact with it for improvement. We now detail each one of these components.

2.1 Interactive Interface

We present ScatterShot as an interactive interface, shown in Figure 2. The interface contains a task description (A_1) and existing in-context examples as demonstrations, presented as input-output pairs (A_2). These pairs are color-encoded based on the text editing distance, with the spans deleted from the input in red, and the spans added in green. Both the description and demonstrations are editable, and are automatically translated into an LLM prompt (Figure 2B) with the task description, demonstrations in the format » `[example input] => [example output]`, and a candidate input for the LLM[1] to transform into an output.

Below the existing examples, ScatterShot proposes a batch of 5 *candidate* inputs sampled from the unlabeled dataset, with outputs computed with the current version of the in-context function (A_3), using the prompt in Figure 2B. Users then verify the candidates and provide feedback (A_3), editing outputs to fix mistakes when needed (*e.g.,* changing from "Thanksgiving == 2000-11-25" to "Thanksgiving == 1999-11-25", A_4), and adding or removing examples to the few-shot examples for in-context learning (A_5). In addition to saving annotation time, LLM-generated outputs help users assess the quality of the current version of the in-context function. For example, if all LLM outputs are correct for a few consecutive batches, it is likely that the existing few-shot examples cover the patterns in the unlabeled data, and thus labeling can stop.

The interface is task-agnostic and can be used whenever users want to learn one-on-one text mapping between text inputs and outputs. This format is flexible, encompassing both classification tasks (where the output is just the class name) and generation tasks

[1]All of our studies and experiments are run on GPT-3 [7], https://beta.openai.com/

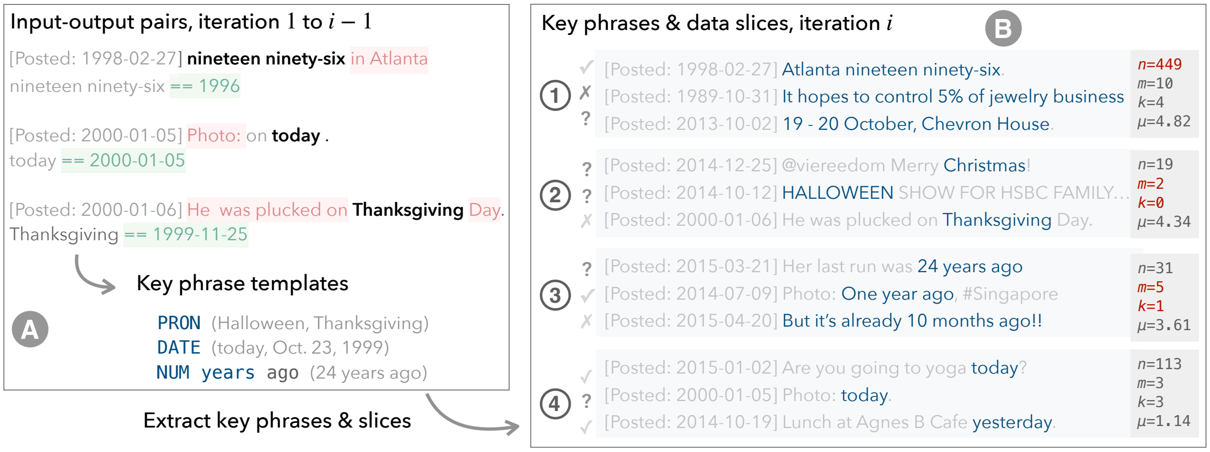

Figure 3: An overview of ScatterShot's slice-based sampling. We use the data status from 1 to $i-1$-th iteration to perform sampling for the i-th iteration. As shown in (A), we use the already sampled input-output pairs to extract *templates* for *task-specific* key phrases. We use these templates to extract key phrases for each unlabeled input, which are the blue highlights in (B). For example, `PRON` helps extract "Christmas" from "@virreedom Merry Christmas!". We run Agglomerative Clustering on the sentence embedding of these key phrases to find task-specific data slices, which contain both not-yet labeled examples (marked with "?") as well as those that have been sampled ("✓" for correctly predicted in previous iterations and "✗ " for incorrect predictions). We rank these slices by an award function μ based on slice size, estimated model performance, and sample frequency, and draw samples from the top clusters.

like summarization, though the color encoding would be most effective for text transformation tasks where the edits from inputs to outputs are worth highlighting. For example, Figure 5 shows how the same interface is used for another question-answer pair rewriting task. ScatterShot can be easily invoked in a Jupyter Notebook, and therefore can support users' natural workflows.

2.2 Slice-based Sampling

2.2.1 *Identifying patterns with key phrase clustering.*

To help users explore both *common* and *less common* patterns, we need a way to partition the unlabeled input examples. While there are a variety of task-agnostic distance metrics that could be used for clustering (*e.g.,* cosine similarity of sentence embeddings [43]), our preliminary exploration indicated that these are typically too coarse when applied to *specific* tasks. For example, using the embeddings from Reimers and Gurevych [43], "Took a photo today." is closer to "Saw a photo on Flickr." (similarity = 0.56) than to "Are you going to yoga class today?" (similarity = 0.30). While this may make sense in the abstract, it does not correspond to how we would want to slice examples for the temporal extraction task in Figure 1, where date references "today" are more important than subject matter ("photos" vs "yoga class"). Thus, we propose a *task-specific* clustering method based on key phrases as explained below.

Detecting key phrases in demonstrations. While key phrase extraction in general may require domain knowledge [8, 42, 65], for text transformation we can leverage the signal present in the relationships between input and output, *i.e.,* in which parts of the input are perturbed or retained. For example, "today" is retained in the output of both "Took a photo today." and "Are you going to yoga class today?" (among many other samples), and thus it is probably a key phrase. Formally, given a labeled, positive example, *i.e.,* a pair of original and perturbed sentences $f(x) => y$, we extract as key phrases either the unmodified parts of x when most of x is changed (Levenshtein edit distance $d(x, y) \geq 0.5$, as is the case with the "today" examples above), or the modified parts when most of x remain unchanged.

Applying key phrases to unlabeled inputs. Applying key phrases naively with an exact match would yield low coverage in the unlabeled data (especially for larger phrases). To get more coverage, at each iteration, we generalize key phrases extracted from labeled demonstrations into *templates* with combinations of tokens, lemmas, and part-of-speech tags [66, 69], *e.g.,*"today" is expanded into `today`, `NOUN`, and `DATE`. We then select representative templates with a greedy weighted set coverage algorithm based on their specificity and the number of inputs they cover [59]. Example templates at various abstraction levels are shown in Figure 3A.

Key phrase clustering. We define the distance between two inputs as the minimum cosine distance between the sentence embeddings [43] *of their key phrases*, and use agglomerative clustering [33] to recursively merge pairs of clusters in the unlabeled data. We set the number of clusters to 20 (chosen empirically in Section 3), and aggregate all clusters with < 10 examples into a single "outlier" cluster (Figure 3B_1). Note that we recompute clusters in every iteration, and thus the outlier cluster tends to shrink as the user interacts with the system. Figure 3B contains various examples of discovered clusters.

Note that as a result of the weighted coverage selection, the templates — and thereby the extracted key phrases — are dynamically changing, and will eventually become more dominant in the sampling procedure: when the few-shot set contains only a few

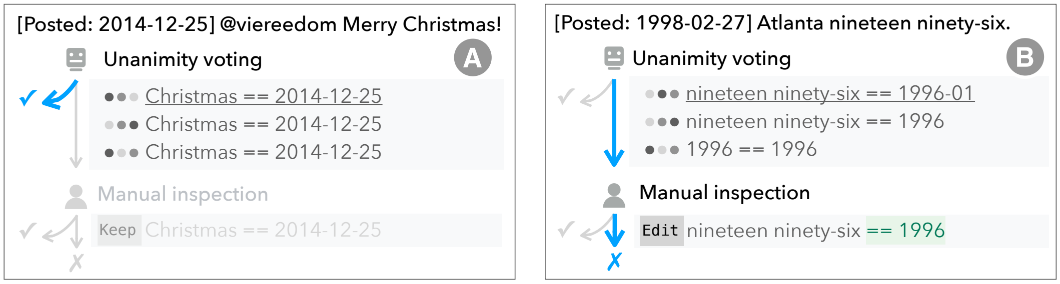

Figure 4: An illustration of ScatterShot's two-step correctness estimation. When the in-context function demonstrates reasonable quality in the last two iterations, we first employ *unanimity voting, i.e.,* we use three different orderings of in-context examples (noted with the three dots with different grey shades) to get three outputs, and say the function is correct if all the outputs are the same, without showing the input to the human (A). When the outputs are different, we return the one with the highest output probability for *user inspection* (underlined), in which manual editing would imply that the function is wrong (B).

(e.g., 3) seeding examples, the templates might be biased or even non-existent, most examples will just use the full sentences as key phrases, making it similar to vanilla clustering on full examples. However, as we add more examples, the templates will be more balanced and eventually stabilize, at which point the clustering can rely more on the extracted key phrases.

2.2.2 Selecting slices for exploration.

We want to explore the identified slices in an efficient way, avoiding slices already "solved," and making the user discovers any unexplored patterns. We take inspiration from the UCB algorithm [4], and use an upper bound estimate of the error of our function in each slice as part of the "reward" for sampling from that slice. Formally, suppose slice c has n examples, m of which are labeled in previous iterations (see the next section for "labeling" details). Further, suppose that out of the m previously labeled examples, the current function is correct on k.[2] The reward of drawing from slice c at iteration i is then given by:

$$\mu_{i,c} = \underbrace{(1-\frac{k}{m})}_{\text{Error Rate}} \cdot \underbrace{\ln n}_{\text{Size}} + \underbrace{\sqrt{\frac{\ln i}{m}}}_{\text{Sample Rarity}}$$

In other words, we prioritize *large slices* ($\ln n$), *low performance* ($1 - k/m$), and slices that have not been sampled many times ($\sqrt{\ln i/m}$, which would give higher weights to clusters with smaller m as the iteration i progresses). Thus, we avoid wasting annotation effort on slices that are already "solved", but keep drawing from slices we can't yet deal with and slices we have not yet explored.

Figure 3B shows four data slices in temporal extraction ranked by reward μ. ① is the "outlier" cluster, where patterns are not yet apparent. This slice still gets prioritized due to its large size (n = 449), even though it has been sampled m = 10, which encourages either higher accuracy or further slicing in follow-up iterations. ② is a slice with holiday-based key phrases. Though the slice is small (n = 19), the LLM failed whenever it was previously sampled (k/m = 0), and thus it currently represents a hard pattern. ③ is a slice with past date references, while ④ is a slice with the common temporal pattern represented by the words "today", "yesterday", and "tomorrow". This last slice has low priority despite being common, since the LLM had perfect accuracy whenever a sample from it was drawn. To maximize diversity (similar to batched active learning [12, 17, 48]), we rank the slices by reward and select one example from each until the batch is filled (in our case, batch size = 5).

2.2.3 Saving user effort with implicit labels and pseudo-labeling.

As mentioned above, our per-slice performance estimation requires *labeled examples*. Unfortunately, we only have firm labels on user-added in-context examples, which may be quite small, especially if users only add a portion of the sampled data. As a result, in-context examples offer limited power for estimation. Although we can modify the interface to collect additional user labels on output correctness, it requires additional interaction that can be cumbersome. To save user effort, we use *implicit labeling*, *i.e.,* we label the LLM output of an example in a batch as correct if the user does not make any changes to the output, even if they do not add it to the in-context demonstration set. Of course, users might ignore model errors if they are frustrated or distracted, but we verified in pilot experiments that users almost always make corrections in the presence of model mistakes (~87% of the time, and the selection method is robust to this small amount of noise). In comparison to explicit labeling, this method requires the bare minimum user interaction, and is easier to integrate into iteration workflows.

Still, implicit labeling requires users to actually *see and interact* with a sample. However, after a certain point in the process, the LLM is correct often enough that many interactions would simply be "accepted" (no changes) by the user. While important for estimating slice accuracy, too much of such interaction might also lead users to overestimate the in-context function quality, and stop the process before they explore the remaining slices. Thus, after we reach a threshold of quality (LLM is correct on 70% of examples in two consecutive rounds), we start leveraging pseudo-labeling with *unanimity voting*, a method inspired by the unanimity principle [23] and Query-by-Committee [34]. Following Lu et al. [32]'s observation that the order of in-context demonstrations can drastically change LLM performance, we form three different prompts by randomly reordering the examples. When the outputs of the prompts agree (*i.e.,* are unanimous), we use that output as a *pseudo-label*, used both for estimating slice accuracy and as a filtering method (*i.e.,* these examples are not shown to the user). Figure 4 illustrates this process,

[2]If an example is in the in-context set, we perform cross-validation and predict its output using the remaining examples.

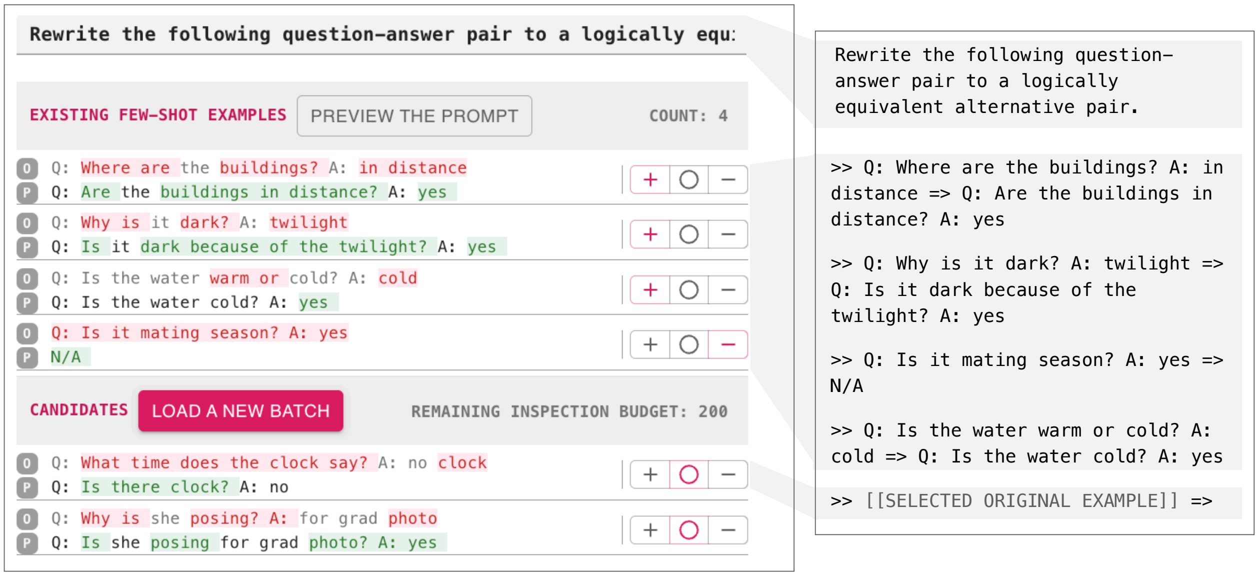

Figure 5: The ScatterShot interface on the question-answer pair implication task.

where "@viereedom Merry Christmas" (A) is pseudo-labeled due to unanimity, and "Atlanta nineteen ninety-six" (B) yields different predictions, and thus is shown to the user for manual inspection.

3 SIMULATION EXPERIMENT: SCATTERSHOT SAMPLING VS. RANDOM SAMPLING

In this section, we measure the effectiveness of slice-based sampling, when compared to random sampling on two text transformation tasks. We use datasets for which we have labels on both tasks, so that we can simulate the labeling process with an oracle at scale, and evaluate the learned function on a held-out portion of each dataset.

3.1 Tasks and Datasets

Temporal expression extraction and normalization. The ***Temporal*** task involves data wrangling [60], where the goal is extracting phrases with temporal expressions from sentences or documents, and normalizing them into a standard format [9]. As shown in Figure 1, these can include absolute or relative dates, and can have different granularity (*e.g.,* exact date vs. year only).

Data. We take the data from [2], containing temporal expression datasets, including TimeBank [41] (news articles) and TweeTime [55] (tweets). We process each dataset into sentences, discarding any date annotations that could not be normalized to the format YYYY-MM-DD (for consistency), and keeping sentences involving absolute dates, dates relative to the document publication date, or no time expressions at all (as the pool for negative examples). This resulted in 491 examples with YYYY-MM-DD outputs, and 369 negative examples with the output N/A. We sample 100 examples randomly from this dataset as a *test set*, and use the remaining examples as our unlabeled pool in the experiment.

Evaluation. Following Chang and Manning [9], we report F1, recall, and accuracy both for the temporal expression extraction and normalization separately.

Question-Answer Pair Implication. For the ***QA-pair*** task, we use ScatterShot to replicate transformation functions from prior work. Given a question-answer (QA) pair, Ribeiro et al. [44] wrote a rule-based system (over 1,000 lines of code[3]) to generate a new QA pair that is *implied* by the original pair, to check whether question answering systems are consistent in their reasoning. We replicate their *logical equivalence* transformation, where the original QA is rewritten to a logically equivalent form, e.g. "Q: What room is this? A: bathroom" is transformed to "Q: Is this a bathroom? A: yes". Despite the heavy engineering, the rule-based system is not able to cover many inputs, and often produces text that does not look fluent or natural. We thus apply in-context learning to this task, and use ScatterShot to select the examples.

Data. We download the input sentences and rule-based implications from Ribeiro et al. [44], and manually inspect and label 1,000 randomly sampled QA pairs (351 rule-based implications were noisy and had to be relabeled). We randomly sample 100 pairs as a test set, and use the remaining pairs as our unlabeled pool in the experiment.

Evaluation. We follow the standard in text generation and report the Rouge-L F scores [28], as well as BLEU-4 [28].

3.2 Procedure and Baseline

We compare *ScatterShot*'s slice-based sampling with a *Random* sampling baseline, which is the most common sampling method used especially in complex tasks, e.g., in text translation [1]. We use GPT-3 as our underlying LLM, with greedy decoding (temperature=0) in both conditions. In each simulation run, we start the process with three random samples (the same for both conditions) of input-output. At every iteration, we compare the ground truth label with the candidate label proposed by the current in-context function. When the labels differ, we add the pair `(input, oracle output)` to the in-context example set, simulating the case where the user

[3] https://github.com/marcotcr/qa_consistency/

Table 1: Quantitative results comparing ScatterShot with the random baseline on *Temporal* and *QA-pair*, averaged over 10 random seeds. ScatterShot outperformed the baseline on all metrics. The significant improvements, measured by student's t-test are marked with *: $p < 0.05$, and **: $p < 0.01$.

Conditions	Extraction			Normalization		
	F1	Precision	Recall	F1	Precision	Recall
Random	73.2 ± 4.0	74.0 ± 3.8	72.9 ± 4.1	66.8 ± 3.2	67.3 ± 3.3	67.0 ± 3.1
ScatterShot	**75.0 ± 2.9**	**75.6± 2.8**	**74.7 ± 2.9**	**70.9 ± 3.4****	**71.3 ± 3.5***	**71.2 ± 3.2****

(a) *Temporal*

Conditions	ROUGE-L	BLEU-4
Rule-based	78.4	66.7
Random	74.3 ± 3.9	65.4 ± 3.5
ScatterShot	**80.0 ± 3.5***	**69.1 ± 3.1***

(b) *QA-pair*

Table 2: Example outputs from transformation functions built in ScatterShot and *Random* condition, and from a rule-based system [44]. ScatterShot functions tend to have better coverage, fluency, and correctness.

Coverage: Transforms more forms of inputs.		
Input	Q: Are there more girls or boys? A: equal	Q: How many hairs does the sheep in front have? A: infinite
Rule-based	✗ *(No generation)*	✗ *(No generation)*
Random	✓ Q: Are the girls and boys equal in number? A: yes	✗ N/A
ScatterShot	✗ Q: Are the girls and boys equal? A: yes	✓ Q: Does the sheep in front have infinite hairs? A: yes
Fluency: Generates outputs that sound natural.		
Input	Q: What make is the phone? A: vtech	Q: What does the woman have on her face? A: headband
Rule-based	✗ Q: Make is the phone vtech? A: yes	✗ Q: Does the woman have on her face headband? A: yes
Random	✓ Q: Is the phone a vtech? A: yes	✓ Q: Does the woman have a headband on her face? A: yes
ScatterShot	✓ Q: Is the phone a vtech? A: yes	✓ Q: Does the woman have a headband on her face? A: yes
Correctness: Produces desired outputs (the new question-pair are logically equivalent to the original pair).		
Input	Q: What monument are they next to? A: unknown	Q: What type of motorcycle is in the picture? A: mountain
Rule-based	✗ Q: Are they next to unknown? A: yes	✗ Q: Is the mountain in the picture ? A: yes
Random	✗ Q: Is the monument unknown? A: yes	✗ Q: Is the mountain type of motorcycle in the picture ? A: yes
ScatterShot	✓ Q: Are they next to an unknown monument? A: yes	✓ Q: Is the motorcycle in the picture a mountain bike? A: yes

corrects a transformation and adds it to the set; Otherwise, the oracle user does not perform any action, simulating cases where the user ignores examples where the current in-context function is correct.

The process is repeated until one of the following stopping conditions is met: (1) the in-context example set contains more than 40 data points (exceeding the LLM maximum context size), (2) The oracle user has been presented with 100 examples (i.e. annotation budget is met), (3) the in-context function provided the correct outputs in five consecutive iterations, or (4) the in-context function's estimated accuracy for all slices of data is ≥ 80%.

We run ten simulation rounds with different random seeds, and report the (averaged) final function performance. We further track the function improvement trajectory over iterations on three randomly selected simulation rounds, by evaluating the intermediate in-context functions after every five examples are added.

3.3 Results

As Table 1 shows, ScatterShot's slice-based sampling outperforms the baseline on both tasks. In *Temporal*, ScatterShot improves the F_1 for date span extraction by around 2 points, and the normalization by 4 points. In *QA-pair*, ScatterShot outperforms *Random* by 6 points on Rouge-L, and even outperforms the heavily engineered rule-based system *used to label most of the evaluation data*, despite needing 40 or fewer in-context examples. Table 2 shows qualitative examples, where ScatterShot outperforms both baselines in terms of coverage, fluency, and correctness. These results point to ScatterShot's potential on saving human efforts in creating fine-grained functions, alleviating the need for handcrafting templates.

Figure 6 shows the trajectory of the in-context function quality as the simulated user adds more examples, for three randomly selected runs. ScatterShot dominates the baseline at almost all points in all runs, with the biggest gaps occurring when the number of in-context examples is small. We see particular gains at $n = 5$, i.e. when the first two examples are added to the seed examples. Our hypothesis (based on qualitative observation) is that ScatterShot consistently selects examples that represent *patterns* not contained in the seed examples, *e.g.,* negative examples (where the outcome is N/A) when all seed examples are positive. While ScatterShot helps users explore most patterns in the unlabeled data as they reach higher n, early gains are especially useful in practice when users have low annotation budgets, *e.g.,* prior work notes users selecting as few as five or ten examples [32, 38].

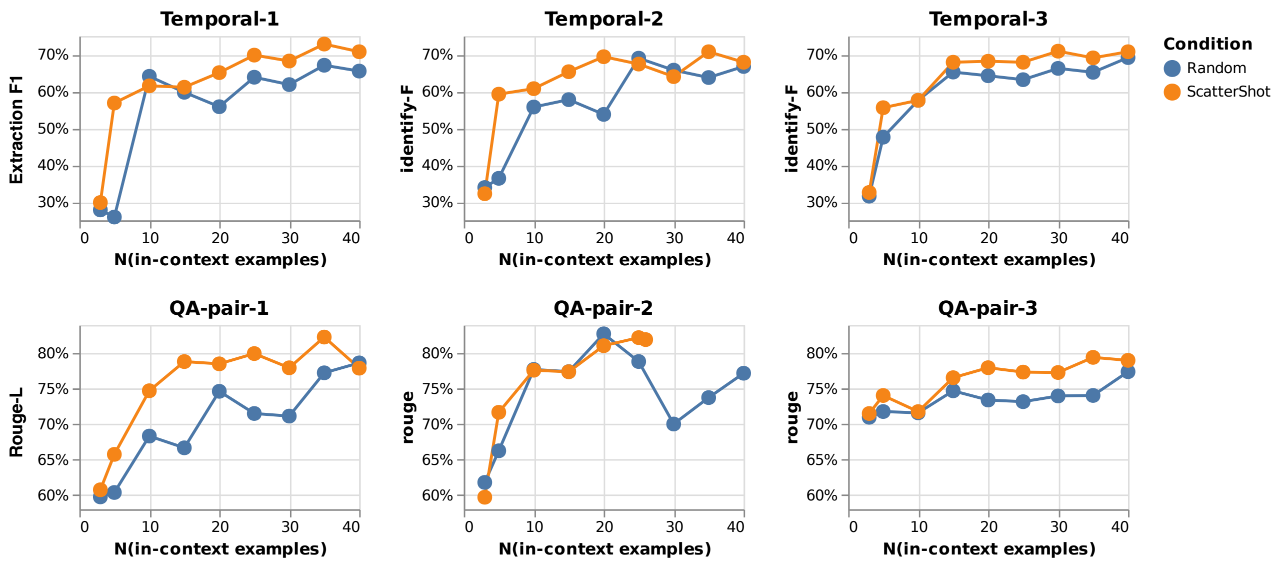

Figure 6: The in-context function performance trajectory, We evaluate the in-context function on the held-out test set every time we add five more examples to the in-context bucket until the stop condition is satisfied. ScatterShot tends to frequently outperform *Random,* and tends to have less performance oscillation. Note that the y-axis is different for *Temporal* and *QA-pair*.

Finally, we observe that ScatterShot is less liable to variance in quality as more examples are added (e.g. in *QA-pair*-2, baseline performance degrades by almost 15 points between $n = 20$ and $n = 30$). These results suggest that besides its interface and interactivity benefits, ScatterShot can improve in-context learning just by virtue of its sample selection function. In order to evaluate the benefits to actual humans, we now turn to a user study.

4 USER STUDY

ScatterShot sampling is effective in simulation, but does it actually aid humans to articulate their desired functions? We conducted a within-subject user study to evaluate whether human users can sense ScatterShot's support in exploring the data space.

4.1 Study Design

Task & Participants. We ran a user study on the *QA-pair* task using the same dataset as Section 3.1, with a split of 900 unlabeled inputs for participants to access, and 100 test examples for evaluating the in-context functions they built. We recruited ten CS graduate student participants (4 females, 6 males) on our CSE department mailing list. Eight of them had previously used GPT-3 and two had heard about it, but none were familiar with the task or ScatterShot. Each participant spent around 60 minutes in the study.

Settings & Conditions. In order to isolate the effect of the different components in ScatterShot, we have two ablation settings in addition to our method: (1) ***Manual***, where participants manually craft prompts without any help from ScatterShot, which is the de-facto *status-quo* of practitioners creating their own in-context learning examples. (2) ***Random***, where participants use the ScatterShot interface with slice-based sampling disabled, *i.e.,* they review randomly selected examples. This condition still has the benefit of an interactive interface, and uses the intermediate in-context functions to suggest outputs and pseudo-label. (3) ***ScatterShot***, where participants have access to ScatterShot, fully featured.

Every participant interacted with every setting in sequence and in a cumulative manner, *i.e.,* the in-context demonstrations gathered in one setting carry over to the next, and we measured the *additional* benefit of moving to the next setting. We divided the participants into two groups, such that in one group the sequence is *Manual* → *Random* → *ScatterShot* (***M-R-S***), while in the other it is *Manual* → *ScatterShot* → *Random* (***M-S-R***). *M-R-S* represents a condition where participants are gradually exposed to more features, such that the step-wise gain maps directly to the benefit of the new feature, while *M-S-R* serves as the counterbalanced condition that combats the learning effect and the natural impact of accumulating examples on function qualities.

Study Procedure. We designed our hour-long study to be self-contained in a Jupyter Notebook,[4] and one of the authors was present in all studies to ensure that participants understood the task and to answer any questions.

Participants were first introduced to the basic concepts of LLM (GPT-3), in-context example construction, and the study task. Then, we randomly assigned the participants to one of the two conditions (*M-R-S* or *M-S-R*), and they completed the task by going through the three conditions in the assigned order. Participants were not instructed on the difference between *ScatterShot* and *Random*, and were instead told that "these two selection methods are randomly ordered, and one is not necessarily better than another."

In each step (setting), participants were told to inspect the inputs and current function outputs (available in *ScatterShot* and *Random*), fix the erroneous outputs, and add demonstrations (input-output pairs) to the in-context example bucket if they believed the data

[4]The full user study instructions, as well as the detailed exit survey, are in https://github.com/tongshuangwu/scattershot

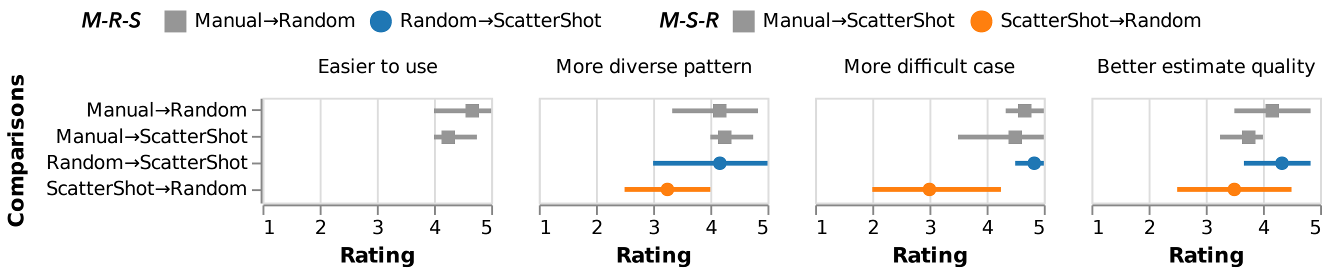

Figure 7: Participants' subjective ratings on their perceived differences between different settings as they switch between them. We use the rectangle to represent when participants first move from *Manual* (Step 1) to the ScatterShot interface (either *ScatterShot* or *Random*, Step 2), and circles to represent switches between ScatterShot interfaces, from one sampling method to the next (Step 2 to 3). Participants strongly preferred the ScatterShot interaction to manual example annotation, and felt they found more diverse patterns and difficult cases in *ScatterShot* than *Random* (*Random→ScatterShot*, blue). In contrast, people in the reversed condition did not find *Random* more useful than *ScatterShot* (orange).

would add additional value, *e.g.,* instances where the current context function fails, as well as diverse input or output patterns. They were asked to iterate within the step until they were satisfied with the in-context function at hand, or accumulated 40 examples. To prevent them from stopping too early, we also asked them to run at least three batches (*i.e.,* see 10-15 examples).[5] Afterwards, participants completed an exit survey and a semi-structured interview, where they rated their perceived experience in each of the two consecutive steps. These questions concerned their perceived input/output pattern diversity, the example difficulty, and their confidence in estimating in-context function quality.

Collected Data. We observed and analyzed three sets of data. First, to quantify the change in **function quality**, we saved participants' in-context examples per step, and applied them to the held-out test set. Here, besides the absolute numbers as in Section 3, we calculated the *difference* in performance between two consecutive steps to see if adding (or, in the case of *M-S-R*, removing) ScatterShot features impacted the quality of examples participants submitted. Second, to assess participants' **self-perceived experience**, we used a standard five-point Likert Scale [27] to collect their perceived step-wise differences. Third, to track participants' **annotation trajectories**, we logged their clickstreams in all the steps. This included both the number of examples they examined per step, the edits they made, and the number of examples they added.

4.2 Results

The ScatterShot interface made it easier to iterate on in-context examples. As shown in Figure 7, participants' found moving from *Manual* (Step 1) to a ScatterShot interface (Step 2) beneficial, regardless of the sampling setting. In particular, they found that the interface made it easier and more intuitive to construct the few-shot examples. (**Easier to use** in Figure 7, 4.7 ± 0.7 for *Manual→Random* and 4.2 ± 0.4 for *Manual→ScatterShot*). Users liked the fact that ScatterShot offers sample inputs (rather than having to go through the dataset on their own), and the that the interface provides easy access to all the existing in-context examples, allowing for fast back-and-forth iteration. For example, one participant (P7) kept revisiting their examples, and removed some earlier examples that they thought were less useful as they became more familiar with the unlabeled input space.

As part of the interface, LLM-generated outputs helped participants craft examples more efficiently, *e.g.,* P6 comments that *"it is less work to make edits than starting from scratch."* Somewhat surprisingly, LLM-generated outputs also improved *output diversity*, *i.e.,* users considered more diverse output patterns. For example, P10 commented that they were *"pleasantly surprised by the LLM's clever output in several cases,"* and that they would not have thought about transformations such as "Q: Is there more than 1 boy? A: no" → "Q: Is there no more than 1 boy? A: yes", which they added to their set of in-context examples. The observation is consistent with prior work showing AI-induced creativity gains [62]. We note that actual user behavior here differs from our simulation setup, where we assumed human users would *only* add new examples when the LLM output was wrong.

Participants' perceptions matched ScatterShot's slice-based sampling design goals: more diverse and more challenging patterns. As shown in Figure 7, participants in *M-R-S* clearly noticed the improvement moving from *Random→ScatterShot* (4.2 ± 1.2 for **more diverse patterns** and 4.8 ± 0.4 for **more difficult case**), whereas most users in *M-S-R* did not report improvements from *ScatterShot→Random*. Qualitative results confirm this, *e.g.,* P7 in *M-R-S* commented: *"Step 2 (*Random*) provided me with some worthy examples, but much less than Step 3 (*ScatterShot*). I went through several rounds of pretty similar examples, thinking the function is behaving quite decently, and didn't realize the function needed more diverse and edge cases until I reached Step 3."* P9 in *M-R-S* was also happy that *ScatterShot* helped them explore beyond typical patterns. In contrast, P10 in *M-S-R* reflected that their exploration seemed to have *"quickly saturated in Step 3"* (*Random*).

Despite not being given details, seven participants discerned the goals behind ScatterShot's sampling method by interacting with it. For example, P2 described it as *"sample for additional variation based on the patterns in existing examples, and also sample for examples similar to previous error mistakes to track whether the function*

[5] In *Manual*, this meant looking at three random batches of unlabeled data in the Jupyter notebook.

Table 3: The performances of participants' in-context functions after each step. +/- represents the average performance change compared to the prior step, whereas the number in the parentheses are the absolute performances. *M-R-S* participants were able to keep adding useful examples, whereas *M-S-R* participants *decreased* the function performance by 0.6 in Step three (*ScatterShot*→*Random*), indicating that these efforts were wasted.

Condition	Step 1	Step 2	Step 3
M-R-S	/ (59.3)	+17.4 (74.7)	**+3.2** (77.8)
M-S-R	/ (61.8)	**+18.1** (75.4)	-0.4 (74.9)

(a) ROUGE-L

Condition	Step 1	→ Step 2	→ Step 3
M-R-S	/ (63.9)	**+10.1** (74.0)	**+3.1** (76.9)
M-S-R	/ (65.3)	+8.9 (74.2)	-0.6 (73.6)

(b) BLEU-4

has been corrected." Two participants in *M-S-R* noticed that *Random* presented fewer mistakes, but attributed it to the increasing number of in-context examples (P5: *"It's getting more correct, but I would expect it given that I have annotated more examples"*). After we explained the selection methods at the end, some users noted that understanding the methods would have helped them better calibrate their estimates of the learned function quality over time.

ScatterShot helped participants explore the input space more holistically, and build better in-context functions. The perceived data difficulty and diversity encouraged participants to iterate more on their in-context examples. When looking at the number of in-context examples added in each setting, participants added 40% more examples in *ScatterShot* than *Random* when *ScatterShot* came after (*M-R-S*), and 20% *fewer* examples in *Random* when *Random* came after (*M-S-R*), *i.e.,* they stopped much earlier when *Random* came after *ScatterShot*. These additional examples are not only a result of more inspection effort (on average, participants in *ScatterShot* reviewed 20% more samples), but also that each batch in *ScatterShot* was more likely to contain a good in-context example — participants added 81% of the examples they inspected in *ScatterShot*, but only 48% of the examples in *Random*.

We report the *quality* of the resulting in-context function on the held-out set in Table 3, and note that *Random*→*ScatterShot* consistently increases performance, while *ScatterShot*→*Random* consistently *decrease* performance despite adding more in-context examples, which is in line with our simulation results.

ScatterShot helped participants estimate function quality and "debug" their example set. As expected, participants estimated their in-context function quality based on the candidate examples they reviewed. For example, P5 (*M-S-R*) tracked the function convergence: *"I made mental notes on the LLM errors and hypothesized what types of examples were missing. For example, I noticed the model was wrong on* `N/A` *questions at first, but later got it right."* Participants in *M-R-S* seemed slightly more satisfied with their estimation, with 4.2 ± 0.9 in *Manual*→*Random* and then further 4.3 ± 0.7 *Random*→*ScatterShot*. P7 commented that *"Step 2 showed me the function is quite smart on patterns it has already seen and has high precision, and Step 3 showed me there are more patterns and it has low recall"*. P2 further reflected that *Random*'s sampling *"created a false impression of convergence, when the function still had various blind spots."* The interactive process also helped participants debug their example sets, *e.g.,* P4 saw big performance drops (4/5 to 1/5 accuracy) on two consecutive batches, which led them to remove in-context examples that were hurting performance.

Participants in *M-S-R* gave slightly lower ratings on their estimates. Qualitatively, the fact that *ScatterShot* prioritized potential mistakes seemed to discourage users, *e.g.,* P3 noted they were driven into *"an endless blackhole of errors,"* after which a round of repetitive patterns in *Random* was hard to make sense of. Once again, this could have been mitigated by *explaining* the sampling strategy to the users, and explicitly displaying the slice accuracy estimates ScatterShot keeps track of.

5 DISCUSSION

In this work, we design a human-LLM collaboration mechanism in ScatterShot to help humans craft diverse and informative in-context learning examples. By iteratively identifying data slices, sampling from low-performance or unseen slices, and providing best-guess outputs for the sampled examples, ScatterShot not only helps the collection of informative in-context examples, but also supports users in exploring the input space and assessing the function quality. At its core, ScatterShot is built on three concepts: data slicing and sampling, iterative human-model interaction, and collaborative human-model labeling. We now discuss challenges and potential future work for each of these.

Slice-based sampling can increase data space coverage. Our experiments showed that sampling from diverse and difficult data slices improves in-context function performance. Importantly, these slices cannot be surfaced via clustering on task-agnostic embeddings; rather, task-specific features should be considered to group examples, while task-irrelevant noise should be minimized. However, identifying these task-specific features remains a challenge. While effective for our function examples (and many others), keyphrase and template extraction would not generalize to tasks where input and output have little syntactic overlap, *e.g.,* English-French translation, summarization, etc. Future work should look into incorporating more general slicing methods, *e.g.,* asking practitioners for slicing functions [11, 42, 65], automatically detecting blind spots [16, 47], etc.

In addition to data slicing, the sampling algorithm also plays a crucial role in narrowing down the actual slices to sample from. We adapt the UCB algorithm to prioritize slice size, performance, and sample rarity, but there are other interesting dimensions that could be explored. For example, if there are slices that cannot be learned after several rounds of sampling, UCB may be counterproductive and create a biased in-context example set that performs worse on *other slices*, whereas a strategy that penalized or just "gave up" on those slices might produce a better overall function. Moreover, we might want to explore better methods for example ranking *within a slice*.

Interacting with the latest function is essential for in-context learning. In-context learning enables rapid function updates, which are not possible in other current interactions with models (*e.g.,* finetuning often takes long hours, and is often not suitable for interactivity). Allowing users to interact with the most current version of what is being learned helps them track progress, and backtrack when they introduce cascading errors [22]. The setup in ScatterShot is a step in this direction, since users always interact with the latest version of the in-context functions.

While participants *were* making progress with ScatterShot (more than with baselines), some participants felt frustrated by inspecting mistake after mistake, fearing that they would never be able to produce a good enough function. While this is by design (ScatterShot prioritizes potential errors), it might compromise annotators' estimates of the quality of their function, and their motivation for labeling more examples. Thus, we notice the importance of presenting quality metrics to the user and clearly explaining the sampling function so that the right expectations are set. For example, users may perform better mental calibration if they have access to hints like the number of slices that are considered "solved" (e.g., as a progress bar that allows people to zoom into concrete examples grouped by the slice), cross-validation accuracy on in-context examples, etc. Another alternative would be to let users exercise *more control* over which slices are explored, *e.g.,* allowing them to "drill down" or "give up" on specific slices.

Human-AI collaborative labeling for building better functions with respect to better quality *and better task definition.* Essentially, ScatterShot enables human-LLM collaboration on data annotation. In our work, we mostly focused on evaluating the quality benefit of such annotation, but we observed additional interesting gains in bringing people inspiration. In Section 4, we notice that participants can take inspiration from the LLM not only on the input patterns, but also on potential output patterns even though our *QA-pair* task is relatively deterministic in its transformations. Thus, we hypothesize that similar systems supporting human-LLM collaborative labeling could play an important role in helping users iteratively refine their task definition and function behavior during data collection. Prior work has shown that annotation requesters refine their labeling instructions when they see noisy (and therefore unusable) crowdsourced labels on ambiguous examples. However, we have yet to examine how LLMs' suggestions (good or bad) might help users better specify their functions.It would be interesting to systematically analyze and measure users' own distribution shift as the example set expands. Recently, Lee et al. [25] proposes the "retaining rate" of LLM suggestions (in their case, suggested character names subsequently used in novels) as a metric of the usefulness of LLMs for ideation. An analogue to our case would be measuring the appearance of new patterns *data slices* when users use ScatterShot, compared to when they come up with their own patterns.

6 RELATED WORK

6.1 LLMs and In-context Learning

Transformer-based large language models (LLMs) [58] have recently led to large improvements in NLP. Pre-trained on a large amount of unlabeled text data, these models encapsulate rich, general-purpose features of language both syntactically and semantically. These features can help facilitate various downstream applications much more dynamically [31] — rather than having to train a new model for every custom task, users can just customize the model by feeding it natural language **prompts** at run time, like the holiday in the previous section. Such ability to recognize the desired task on-the-fly is called *in-context learning* [7].

The flexible in-context learning intrigues various work to explore designing prompts that can effectively invoke the user desired functionalities [35, 37, 46, 70]. To date, 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 the holiday date translator in Section 1 with a *task description* prompt: "Identify the date for a national holiday in the month/date format." Studies on improving zero-shot prompts typically study the effect of task instructions [15], induce LLM reasoning through task decomposition [63, 67], etc. Zero-shot prompts do not use demonstrative examples and therefore tend to be less performative [7], but writing just the natural language descriptions is lightweight enough that it creates an intuitive natural language interface between humans and the model [64].

In contrast, *few-shot* prompts show the LLM what pattern to follow by feeding it examples of the desired input and output data. As can be seen in Section 1, given examples on "Christmas" and "Halloween", the LLM would produce a reasonable date for "Independence Day". These examples usually follow consistent structures with meaningful prefixes ("Holiday: [name] => Date: [date]"), which helps re-emphasize the desired intent [58]. The quality of few-shot prompts heavily relies on the five to thirty in-context examples that demonstrate the intended behavior [32, 46], and LLMs can only perform in-context learning if it has seen the corresponding distribution or concept [35, 46, 70]. If developers omit corner cases in the few examples they created, the task quality can easily be affected [29]. For example, without a negative example where we denote ineligible inputs with a placeholder output "N/A" ("Holiday: yesterday => Date: N/A"), the LLM would attempt to produce the most plausible "label" even for negative examples — It may try to normalize "yesterday" to a most plausible date even though there is no *holiday*. Our work here tries to help users interactively identify high-quality in-context examples for text transformation. We review the literature on in-context example selection next.

6.2 Effective Example Selection

Prior work has explored selecting effective demonstrations, and has shown that because pre-trained models possess high-level semantic features, sampling or active learning tends to help identify informative examples [56]. In particular, dynamically selecting (retrieving) the most similar demonstrative examples for each given input significantly improves in-context learning performance [10, 46]. However, such retrieval methods require fully labeled datasets as the search space. In contrast, our work studies the scenario where humans craft their personalized in-context functions, and therefore focuses on an unlabeled space.

In the unlabeled search space, prior work has explored effective dataset annotation that can support better in-context learning or few-shot finetuning. These studies strive to allocate annotation budgets to diverse and representative examples through clustering [10] or graph-based search [53]. For example, Su et al. [53] built a similarity graph by computing pairwise distances between input sentences and then iteratively selected and annotated examples based on graph density. They show such selection substantially reduces the annotation cost while achieving high and stable in-context learning performance. Despite being effective, these methods sample examples purely for input diversity. Because our work focuses more on supporting users' interactive function construction, we additionally emphasize *current function quality* in sampling, which helps users track their progress and prioritize improving the current in-context function. Moreover, these prior studies measures diversity with cosine similarities on input sentence embedding [43] which, as we argue in Section 2.2, is not reflective of various tasks [46]. As a workaround, our work focuses on measuring similarities only on the key phrase embeddings, which leads to more intuitive clusters.

On the interactive example selection side, our work is perhaps more similar to some literature in programming-by-demonstration (PBD). For example, Zhang et al. [72] explored effectively selecting examples that can help disambiguate and validate synthesized regular expressions. We share similar motivations that interactively and iteratively suggest 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.

Active Learning. Our work is also similar to the aforementioned, effective annotation work [10, 53] in the sense that its selection method is akin to sampling approaches in active learning [49, 57]. 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. While the previous work is more similar to diversity sampling [48], ours is closer to uncertainty sampling [26], where an active learner queries the instances about which it is least certain how to label. 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 (unanimity voting) which also conveniently serves as a correctness estimation. This voting strategy is also quite relevant to Query-By-Committee [50] where a list of "committee" models trained on the same labeled set vote on the labelings of query candidates. Other work has also been considered directly representing LLM confidence with the average log probability of the entire output [53, 61], an alternative worth comparing against in the future.

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 [49], and therefore can sometimes be imbalanced [40] or less effective than random sampling [20]. Our method will likely share the same limitations, though we leave it to future work to articulate scenarios where ScatterShot is most useful.

6.3 Model-assisted Annotation

ScatterShot can also be seen as offering assistance in data annotation (for context learning). The idea of annotating data with both humans and AI models in the loop has been explored broadly. In this setup, AIs can play various roles [71], *e.g.,* they may generate more examples that mimic difficult patterns [29, 45], select uncertain examples for people to inspect [61], etc. ScatterShot is closer to work encouraging annotators to find model-fooling examples ("adversarial data collection.") [6, 13, 14, 24]. In particular, Bartolo et al. [5] found that in question-answering tasks, models trained on these adversarially collected data can generalize better to more challenging examples. However, because of the overhead of re-training, their analyses were performed *post-hoc*, *i.e.,* they only updated the model offline after collecting a large batch of challenging examples. In contrast, we leverage the advantage of in-context learning, and directly study the dynamic of in-context function update.

The iterative nature also links ScatterShot to earlier work in interactive machine learning (IML) [3, 68]. IML is a typical paradigm that facilitates iterative and exploratory model understanding and update — a system explains to users how the current model makes predictions, and users in turn give feedback back to the model, starting the cycle again. Labeling is one classic type of IML feedback [19, 51]. However, because traditional ML tends to focus much more on the surface features (*e.g.,* count trigrams in a training example without caring its semantic meanings), users find labeling to be not powerful enough, and prefer richer controls like feature selection [3, 39, 52]. Since LLMs have some capability to generalize individual examples more broadly to its semantically similar ones, we believe labeling in in-context learning would be more effective, and we use ScatterShot to reactivate labeling-based IML for in-context learning.

7 CONCLUSION

In this work, we present ScatterShot, an interactive system for building high-quality demonstration sets for in-context learning. ScatterShot helps users find informative input examples in the unlabeled data, annotate them efficiently with the help of the current version of the learned in-context function, and estimate the quality of said function. Results from both a simulation study and a 10-person evaluation show ScatterShot improves in-context function performance, as well as annotator's awareness and handling of diverse patterns. Our findings highlight the importance of data slicing and sampling, iterative human-model interaction, and collaborative human-model labeling, and point to interesting future directions such as AI-assisted task definition refinement, more concrete quality metrics that convey the in-context function progress, etc.

ACKNOWLEDGMENTS

This material is based upon work supported by NSF awards 1901386 and 2040196, ONR grant N00014-21-1-2707, and a gift from the Allen Institute for Artificial Intelligence (AI2). The authors thank the user study participants for their valuable feedback, and anonymous reviewers for helpful discussions and comments.

REFERENCES

[1] Sweta Agrawal, Chunting Zhou, Mike Lewis, Luke Zettlemoyer, and Marjan Ghazvininejad. 2022. In-context Examples Selection for Machine Translation. *ArXiv preprint* abs/2212.02437 (2022). https://arxiv.org/abs/2212.02437

[2] Satya Almasian, Dennis Aumiller, and Michael Gertz. 2021. BERT got a Date: Introducing Transformers to Temporal Tagging. *ArXiv preprint* abs/2109.14927 (2021). https://arxiv.org/abs/2109.14927

[3] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *Ai Magazine* 35, 4 (2014), 105–120.

[4] Peter Auer. 2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* 3, Nov (2002), 397–422.

[5] Max Bartolo, Alastair Roberts, Johannes Welbl, Sebastian Riedel, and Pontus Stenetorp. 2020. Beat the AI: Investigating Adversarial Human Annotation for Reading Comprehension. *Transactions of the Association for Computational Linguistics* 8 (2020), 662–678. https://doi.org/10.1162/tacl_a_00338

[6] Max Bartolo, Tristan Thrush, Sebastian Riedel, Pontus Stenetorp, Robin Jia, and Douwe Kiela. 2022. Models in the Loop: Aiding Crowdworkers with Generative Annotation Assistants. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, United States, 3754–3767. https://doi.org/10.18653/v1/2022.naacl-main.275

[7] 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. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[8] Ángel Alexander Cabrera, Marco Tulio Ribeiro, Bongshin Lee, Rob DeLine, Adam Perer, and Steven M Drucker. 2022. What Did My AI Learn? How Data Scientists Make Sense of Model Behavior. *ACM Transactions on Computer-Human Interaction* (2022).

[9] Angel X. Chang and Christopher Manning. 2012. SUTime: A library for recognizing and normalizing time expressions. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. European Language Resources Association (ELRA), Istanbul, Turkey, 3735–3740. http://www.lrec-conf.org/proceedings/lrec2012/pdf/284_Paper.pdf

[10] Ernie Chang, Xiaoyu Shen, Hui-Syuan Yeh, and Vera Demberg. 2021. On Training Instance Selection for Few-Shot Neural Text Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, Online, 8–13. https://doi.org/10.18653/v1/2021.acl-short.2

[11] Vincent S. Chen, Sen Wu, Alexander J. Ratner, Jen Weng, and Christopher Ré. 2019. Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 9392–9402. https://proceedings.neurips.cc/paper/2019/hash/351869bde8b9d6ad1e3090bd173f600d-Abstract.html

[12] Gui Citovsky, Giulia DeSalvo, Claudio Gentile, Lazaros Karydas, Anand Rajagopalan, Afshin Rostamizadeh, and Sanjiv Kumar. 2021. Batch active learning at scale. *Advances in Neural Information Processing Systems* 34 (2021), 11933–11944.

[13] Emily Dinan, Samuel Humeau, Bharath Chintagunta, and Jason Weston. 2019. Build it Break it Fix it for Dialogue Safety: Robustness from Adversarial Human Attack. 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)*. Association for Computational Linguistics, Hong Kong, China, 4537–4546. https://doi.org/10.18653/v1/D19-1461

[14] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 2368–2378. https://doi.org/10.18653/v1/N19-1246

[15] Avia Efrat and Omer Levy. 2020. The Turking Test: Can Language Models Understand Instructions? *ArXiv preprint* abs/2010.11982 (2020). https://arxiv.org/abs/2010.11982

[16] Sabri Eyuboglu, Maya Varma, Khaled Saab, Jean-Benoit Delbrouck, Christopher Lee-Messer, Jared Dunnmon, James Zou, and Christopher Ré. 2022. Domino: Discovering systematic errors with cross-modal embeddings. *ArXiv preprint* abs/2203.14960 (2022). https://arxiv.org/abs/2203.14960

[17] Yonatan Geifman and Ran El-Yaniv. 2017. Deep active learning over the long tail. *ArXiv preprint* abs/1711.00941 (2017). https://arxiv.org/abs/1711.00941

[18] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. 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)*. Association for Computational Linguistics, New Orleans, Louisiana, 107–112. https://doi.org/10.18653/v1/N18-2017

[19] Florian Heimerl, Steffen Koch, Harald Bosch, and Thomas Ertl. 2012. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (2012), 2839–2848.

[20] Henrik Imberg, Johan Jonasson, and Marina Axelson-Fisk. 2020. Optimal sampling in unbiased active learning. In *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy] (Proceedings of Machine Learning Research, Vol. 108)*, Silvia Chiappa and Roberto Calandra (Eds.). PMLR, 559–569. http://proceedings.mlr.press/v108/imberg20a.html

[21] Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J. Cai. 2022. Prompt-based Prototyping with Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*.

[22] Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J Cai. 2022. PromptMaker: Prompt-based Prototyping with Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–8.

[23] Fereshte Khani, Martin Rinard, and Percy Liang. 2016. Unanimous Prediction for 100% Precision with Application to Learning Semantic Mappings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 952–962. https://doi.org/10.18653/v1/P16-1090

[24] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. 2021. Dynabench: Rethinking Benchmarking in NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 4110–4124. https://doi.org/10.18653/v1/2021.naacl-main.324

[25] Mina Lee, Percy Liang, and Qian Yang. 2022. CoAuthor: Designing a Human-AI Collaborative Writing Dataset for Exploring Language Model Capabilities. *ArXiv preprint* abs/2201.06796 (2022). https://arxiv.org/abs/2201.06796

[26] David D Lewis and William A Gale. 1994. A sequential algorithm for training text classifiers. In *SIGIR'94*. Springer, 3–12.

[27] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[28] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[29] Alisa Liu, Swabha Swayamdipta, Noah A Smith, and Yejin Choi. 2022. Wanli: Worker and ai collaboration for natural language inference dataset creation. *ArXiv preprint* abs/2201.05955 (2022). https://arxiv.org/abs/2201.05955

[30] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What Makes Good In-Context Examples for GPT-3?. In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. Association for Computational Linguistics, Dublin, Ireland and Online, 100–114. https://doi.org/10.18653/v1/2022.deelio-1.10

[31] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ArXiv preprint* abs/2107.13586 (2021). https://arxiv.org/abs/2107.13586

[32] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 8086–8098. https://doi.org/10.18653/v1/2022.acl-long.556

[33] Alena Lukasová. 1979. Hierarchical agglomerative clustering procedure. *Pattern Recognition* 11, 5-6 (1979), 365–381.

[34] Prem Melville and Raymond J. Mooney. 2004. Diverse ensembles for active learning. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004 (ACM International Conference Proceeding Series, Vol. 69)*, Carla E. Brodley (Ed.). ACM. https://doi.org/10.1145/1015330.1015385

[35] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? *ArXiv preprint* abs/2202.12837 (2022). https://arxiv.org/abs/2202.12837

[36] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-Task Generalization via Natural Language Crowdsourcing Instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 3470–3487. https://doi.org/10.18653/v1/2022.acl-long.244

[37] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-Task Generalization via Natural Language Crowdsourcing Instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 3470–3487. https://doi.org/10.18653/v1/2022.acl-long.244

[38] Joon Sung Park, Lindsay Popowski, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2022. Social Simulacra: Creating Populated Prototypes for Social Computing Systems. *ArXiv preprint* abs/2208.04024 (2022). https://arxiv.org/abs/2208.04024

[39] Kayur Patel, James Fogarty, James A Landay, and Beverly Harrison. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 667–676.

[40] Remus Pop and Patric Fulop. 2018. Deep ensemble bayesian active learning: Addressing the mode collapse issue in monte carlo dropout via ensembles. *ArXiv preprint* abs/1811.03897 (2018). https://arxiv.org/abs/1811.03897

[41] James Pustejovsky, Jessica Littman, Roser Saurí, and Marc Verhagen. 2006. Timebank 1.2 documentation. *Event London, no. April* (2006), 6–11.

[42] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 11. NIH Public Access, 269.

[43] Nils Reimers and Iryna Gurevych. 2019. 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)*. Association for Computational Linguistics, Hong Kong, China, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[44] Marco Tulio Ribeiro, Carlos Guestrin, and Sameer Singh. 2019. Are Red Roses Red? Evaluating Consistency of Question-Answering Models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 6174–6184. https://doi.org/10.18653/v1/P19-1621

[45] Marco Tulio Ribeiro and Scott Lundberg. 2022. Adaptive Testing and Debugging of NLP Models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Dublin, Ireland, 3253–3267. https://doi.org/10.18653/v1/2022.acl-long.230

[46] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning To Retrieve Prompts for In-Context Learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, United States, 2655–2671. https://doi.org/10.18653/v1/2022.naacl-main.191

[47] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. 2019. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *ArXiv preprint* abs/1911.08731 (2019). https://arxiv.org/abs/1911.08731

[48] Ozan Sener and Silvio Savarese. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=H1aIuk-RW

[49] Burr Settles. 2009. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

[50] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. 1992. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*. 287–294.

[51] Patrice Simard, David Chickering, Aparna Lakshmiratan, Denis Charles, Léon Bottou, Carlos Garcia Jurado Suarez, David Grangier, Saleema Amershi, Johan Verwey, and Jina Suh. 2014. Ice: enabling non-experts to build models interactively for large-scale lopsided problems. *arXiv preprint arXiv:1409.4814* (2014).

[52] Simone Stumpf, Vidya Rajaram, Lida Li, Weng-Keen Wong, Margaret Burnett, Thomas Dietterich, Erin Sullivan, and Jonathan Herlocker. 2009. Interacting meaningfully with machine learning systems: Three experiments. *International journal of human-computer studies* 67, 8 (2009), 639–662.

[53] Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. *ArXiv preprint* abs/2209.01975 (2022). https://arxiv.org/abs/2209.01975

[54] Ben Swanson, Kory Mathewson, Ben Pietrzak, Sherol Chen, and Monica Dinalescu. 2021. 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*. Association for Computational Linguistics, Online, 244–256. https://doi.org/10.18653/v1/2021.eacl-demos.29

[55] Jeniya Tabassum, Alan Ritter, and Wei Xu. 2016. TweeTime : A Minimally Supervised Method for Recognizing and Normalizing Time Expressions in Twitter. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, 307–318. https://doi.org/10.18653/v1/D16-1030

[56] Alex Tamkin, Dat Nguyen, Salil Deshpande, Jesse Mu, and Noah Goodman. 2022. Active Learning Helps Pretrained Models Learn the Intended Task. *ArXiv preprint* abs/2204.08491 (2022). https://arxiv.org/abs/2204.08491

[57] Toan Tran, Thanh-Toan Do, Ian D. Reid, and Gustavo Carneiro. 2019. Bayesian Generative Active Deep Learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 6295–6304. http://proceedings.mlr.press/v97/tran19a.html

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[59] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.

[60] Gust Verbruggen, Vu Le, and Sumit Gulwani. 2021. Semantic programming by example with pre-trained models. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–25.

[61] Shuohang Wang, Yang Liu, Yichong Xu, Chenguang Zhu, and Michael Zeng. 2021. Want To Reduce Labeling Cost? GPT-3 Can Help. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 4195–4205. https://doi.org/10.18653/v1/2021.findings-emnlp.354

[62] Yunlong Wang, Priyadarshini Venkatesh, and Brian Y Lim. 2022. Interpretable Directed Diversity: Leveraging Model Explanations for Iterative Crowd Ideation. In *CHI Conference on Human Factors in Computing Systems*. 1–28.

[63] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *ArXiv preprint* abs/2201.11903 (2022). https://arxiv.org/abs/2201.11903

[64] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–10.

[65] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2019. Errudite: Scalable, Reproducible, and Testable Error Analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 747–763. https://doi.org/10.18653/v1/P19-1073

[66] Tongshuang Wu, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2021. 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)*. Association for Computational Linguistics, Online, 6707–6723. https://doi.org/10.18653/v1/2021.acl-long.523

[67] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. 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* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 385, 22 pages. https://doi.org/10.1145/3491102.3517582

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

[69] Tongshuang Wu, Kanit Wongsuphasawat, Donghao Ren, Kayur Patel, and Chris DuBois. 2020. Tempura: Query Analysis with Structural Templates. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, 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 (Eds.). ACM, 1–12. https://doi.org/10.1145/3313831.3376451

[70] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *ArXiv preprint* abs/2111.02080 (2021). https://arxiv.org/abs/2111.02080

[71] Zhilin Yang, Saizheng Zhang, Jack Urbanek, Will Feng, Alexander H. Miller, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Mastering the Dungeon: Grounded Language Learning by Mechanical Turker Descent. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJ-C6JbRW

[72] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. 2020. Interactive Program Synthesis by Augmented Examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 627–648.