Review: Improving workflow, turning errors into facts, and achieving true human-like reasoning

Van Trinh 2024

Data annotation is a key step in training AI models. With a focus on improving labeling, models achieve high levels of accuracy and precision in performing their intended tasks. On the other hand, models progressively degrade and correcting their mistakes may require much more time and resources than initially intended. Below is a review based on published articles on some of the many tools available for improving the labeling process and thus enhancing AI model performance.

I. Improving workflow

Human-assisted computation algorithms, data labeling in particular, increase performance when there's a focus on workflow design. Without it, there is an increased risk of error and the quality of work in the labeling process may be affected. Below are several successful design patterns studied in the published literature, and how they may be applied to AI training projects.

1. Implement a multi-stage process

There are multiple ways to conduct a multi-stage workflow, and one of which is Find-Fix-Verify. Find-Fix-Verify is a design pattern used by Soylent, a tool designed by MIT researchers to be embedded in Microsoft Word to shorten text while preserving the content of the text. When a user selects an area of text in Word and presses a button in the Soylent ribbon tab, Soylent launches a series of tasks on the Amazon Mechanical Turk platform. This platform recruits human workers to do the tasks and subsequently outputs the results back to Soylent, which then computes the combination of trimmings by humans that most closely match the desired text length given by the user (Bernstein et al., 2010).

Find-Fix-Verify splits the tasks into three stages, completed by three different groups of workers. “Find” workers identify areas to shorten in the text, “Fix” workers suggest ideas for revisions, and “Verify” workers vote to choose option (Figure 1) (Bernstein et al., 2010). This design has more accuracy compared to letting an individual worker perform all three stages for each task.

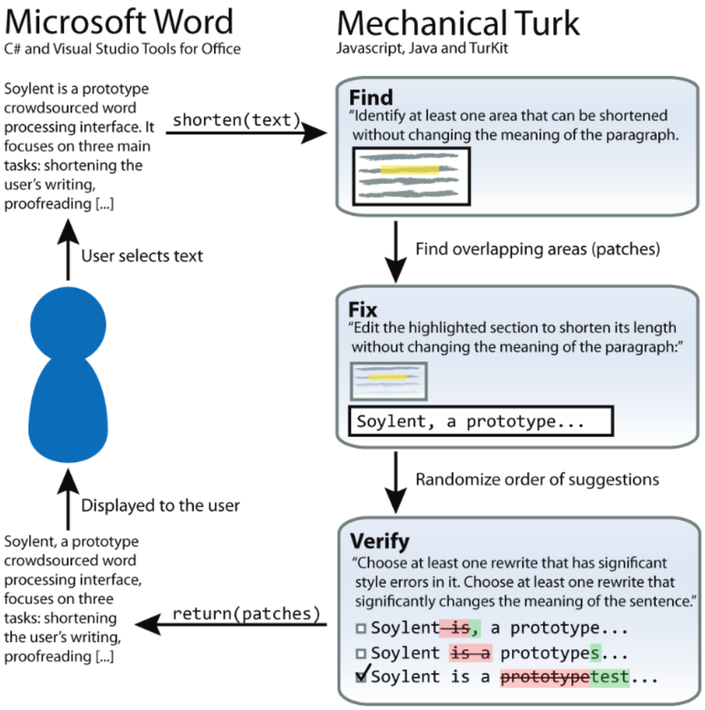

Microsoft Word
C# and Visual Studio Tools for Office
Soylent is a prototype crowdsourced word processing interface. It focuses on three main tasks: shortening the user's writing, proofreading [...]
shorten(text)
User selects text
Displayed to the user
Soylent, a prototype crowdsourced word processing interface, focuses on three tasks: shortening the user's writing, proofreading [...]
return(patches)
Mechanical Turk
Javascript, Java and TurKit
Find
"Identify at least one area that can be shortened without changing the meaning of the paragraph.
Find overlapping areas (patches)
Fix
"Edit the highlighted section to shorten its length without changing the meaning of the paragraph:"
Soylent, a prototype...
Randomize order of suggestions
Verify
"Choose at least one rewrite that has significant style errors in it. Choose at least one rewrite that significantly changes the meaning of the sentence."
□ Soylent ~~is~~, a prototype...
□ Soylent ~~is a~~ prototypes...
✓ Soylent is a ~~prototype~~test...

Figure 1. Find-Fix-Verify workflow identifies areas to change, suggests revisions, and votes to choose the best option. (Bernstein et al., 2010)

2. Breaking down tasks: Partition-Map-Reduce

Partition-Map-Reduce pattern suggests that instead of viewing workers separately, you can also view each worker as node in a coupled distributed computing system, in which each worker is analogous to a computer processor that can solve a task requiring human intelligence (Kittur et al., 2011). Therefore, we can apply distributed computing solutions to managing human-assisted computation. MapReduce was inspired by functional programming languages in which a large array of data is processed in parallel through a two-step process: first, key/value pairs are each processed to generate a set of intermediate key/value pairs (the "Map" phase). Next, values with the exact same intermediate keys are merged (the "Reduce" phase). Map and Reduce processes may be nested. As a result, tasks that do not depend on each other are broken down, local improvements can be made with more ease compared to keeping tasks in chunks, and the process is more streamlined.

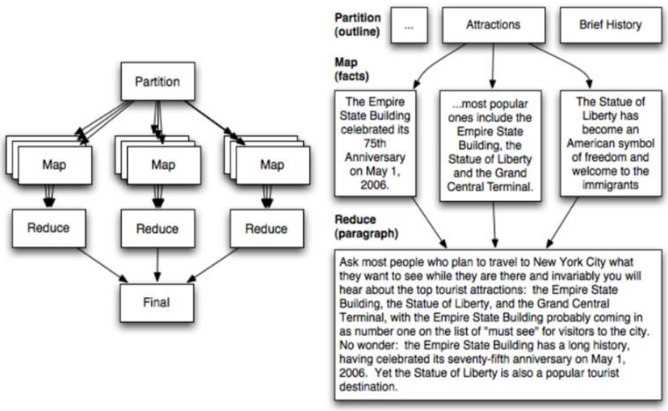

Partition
Map
Map
Map
Reduce
Reduce
Reduce
Final
Partition (outline)
...
Attractions
Brief History
Map (facts)
The Empire State Building celebrated its 75th Anniversary on May 1, 2006.
...most popular ones include the Empire State Building, the Statue of Liberty and the Grand Central Terminal.
The Statue of Liberty has become an American symbol of freedom and welcome to the immigrants
Reduce (paragraph)
Ask most people who plan to travel to New York City what they want to see while they are there and invariably you will hear about the top tourist attractions: the Empire State Building, the Statue of Liberty, and the Grand Central Terminal, with the Empire State Building probably coming in as number one on the list of "must see" for visitors to the city. No wonder: the Empire State Building has a long history, having celebrated its seventy-fifth anniversary on May 1, 2006. Yet the Statue of Liberty is also a popular tourist destination.

Figure 2. Partition-Map-Reduce workflow design (Kittur et al., 2011).

II. Turning errors into facts.

Data labeling, the process of tagging or marking datasets, can be approached through the lens of classification - dividing data with different labels or classes. Binary classification models predict a binary outcome, that is, one of two classes. For example, in a yes or no question, there are only two outcomes, “yes” and “no”. Multi-class classification models predict one class from three or more classes, used to perform text and image disambiguation, recognition, identification, and various other purposes.

In data annotation projects, many tasks compare between responses A and B to output a rating, such as: “A and B are the same”, “A is slightly better than B”, “A is much better than B”, etc. Each rating category can be viewed as a class. Thus, a multi-class classification model can be used to predict which class of rating is the most likely for a certain task. A recent study by researchers at Stanford and Yahoo Labs (Krishna et al., 2016) proposes a class-optimized approach for multi-class classification built upon a binary classification technique that reduces data labeling speed by an order of magnitude while still maintaining the same quality. First, let’s discuss the class-optimized approach for multi-class classification, a simple method that may be applied immediately to data annotation projects. Afterwards, we’ll go into the details of the binary classification technique.

Theoretically, all multi-class classification can be broken down into a series of binary verifications. For example, if there are N classes, there can be N binary questions of whether an item is in each class. Given a list of items, one item can be classified at a time by asking each class or bucket a yes-or-no question. After every iteration, all items classified for a particular class are removed. The rest of the items can be classified for the next class. The order in which classes are considered should not matter when all the classes contain an equal number of items. A simple baseline approach would choose a class at random and attempt to detect all items for that class first. However, if classes end up with uneven sizes, such a method would require more time. Consider the case where items are to be classified into 5 classes. The sizes of the classes are initially unknown, but at the end of the process, one class has 1000 items, while all others have 10 items each. In the worst case, if items that belong to the class that ends up having 1000 examples are classified last, each item would go through the interface 5 times, once for every class, for a total of 5000 times (Figure 3). Instead, if all items that belong to this class are detected first, those 1000 images would only go through the interface once. With this intuition, the authors propose a class-optimized approach that classifies the class with the most items first. The number of items classified at every iteration is maximized, reducing the total number of binary verifications. The authors demonstrate this approach by classifying a dataset of 2,000 images with 10 categories, where each category contains 100 to 250 examples. The class-optimized approach completes the task at a speed of 11,700 seconds, while the baseline approach does so in 12,342 seconds, a difference that will likely be widened as the dataset grows larger. One caveat is that in the study, the identity of the class with the most items is known in advance. For a dataset with unknown sizes, it might be good to do a run to first gauge the frequency of classes, which can then be used to inform future runs.

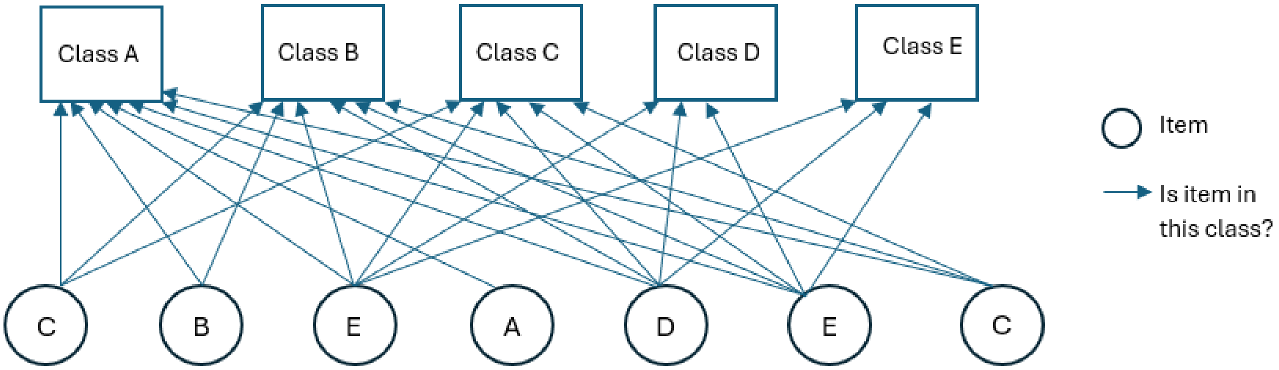


Figure 3. Each item queries each class until it finds its own class and is removed. Class sizes are initially unknown. If items in the largest class are classified last, all of them must query all other classes. Classifying the class with the most items first should reduce the total number of queries.

Binary classification technique proposed by Krishna et al. (2016) can potentially be applied to future data annotation projects. To reduce cost, many previous studies have examined the tradeoff between speed and accuracy in the annotation process. Some methods estimate speed and accuracy to jointly model errors, other methods vary speed and accuracy to calculate the tradeoff between the two, and another predicts the redundancy of non-expert labels to match the level of expert labels (Wah et al., 2014; Branson et al., 2010; Wah et al., 2011). Below is a simple yet powerful method to reducing speed while still maintaining the highest quality.

Krishna et al. (2016) first discusses how current methods of evaluating annotation work usually punish all errors. Meanwhile, in their study, errors are fully expected and even used to predict the correct outcomes. Their method encourages a margin of human error in the interface that is later rectified by inferring true labels algorithmically.

In the past, studies have explored different ways to use workers in a model: workers themselves have been modeled as projections into open space, their skills have been modeled in a generative method, and work quality has been modeled in an unsupervised method (Whitehill et al., 2009; Zhou et al., 2012; Ipeirotis et al., 2010). This study proposes a method not requiring many algorithms yet allowing the prediction of correct outcomes from errors: creating a normal probability distribution for each keypress and then modeling the keypress latency to predict the correct labels.

In the study, each worker sees an image that is displayed for 400 ms alongside a binary question. The worker presses the space bar when the answer is “yes” to the question. The worker is only given 400 ms to react, so errors are expected. There is a delay between when the image appears and when the worker presses the space bar. Given a keypress, this delay is unknown. Thus, each keypress is treated as a normal probability distribution within 400 ms before the keypress (Figure 4).

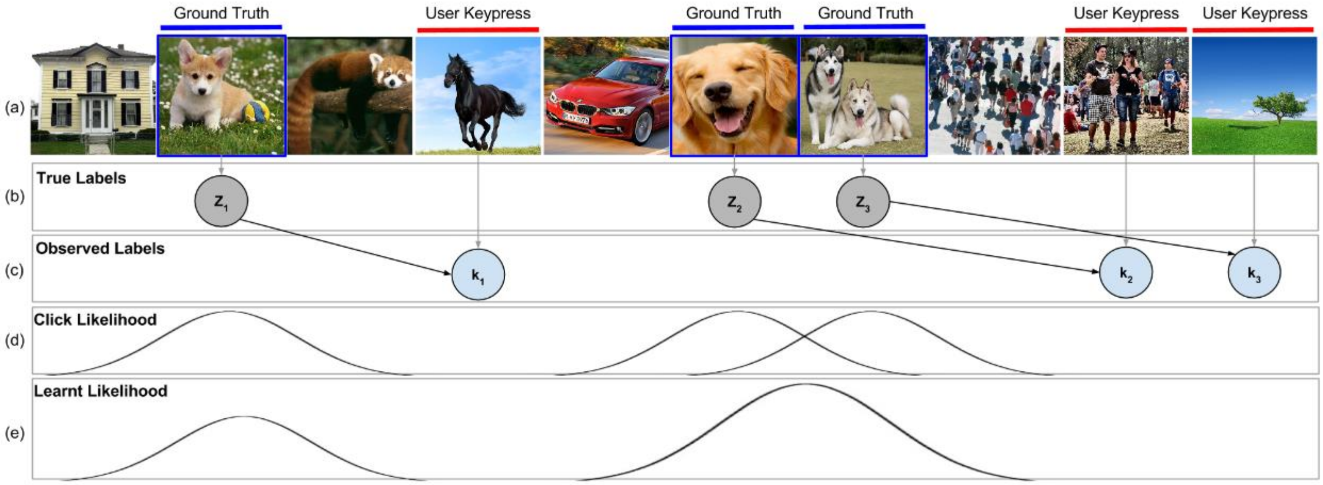


Figure 4. (a) Images are shown, and workers react whenever they see a dog. (b) The ground truth dog images are the true labels. (c) The workers’ keypresses occur several images after the dog images have already passed. These keypresses are recorded as the observed labels. (d) Each keypress is modeled as a delayed Gaussian to predict (e) the probability of an image containing a dog from these observed labels. (Krishna et al., 2016)

Because of the worker’s reaction delay, the data from each worker has much uncertainty. The same set of images is shown to multiple workers in random orders, and independent sets of keypresses are collected. This randomization allows for unbiased estimates of probabilities and cleaner signal in aggregate during majority voting, a previously studied method that this new method is compared to.

The goal is to translate keypresses that may be erroneous into the identifications of positive items. To do so, Krishna et al. (2016) calculates the probability that a particular image is correctly identified given a worker’s keypress. Next, the mean and the variance of the distribution of each keypress are used to train a model. The probabilities across several workers are summed up to calculate the total probability for each image. All images above a certain threshold are set as “positive”, that is, correctly identified. This threshold is a hyperparameter in the model and can be tuned.

Here are the model details. A set of items $I = \{I_1, \ldots, I_n\}$ are sent to W workers in a different random order for each worker. From each worker w, a set of keypresses $C^w = \{C_1^w, C_2^w, \ldots, C_k^w\}$ is collected, where $w \in W$ and k is the total number of keypresses from worker w. The aim is to calculate the probability of an item being correctly identified given a set of keypresses, $P(I_i \mid C^w)$.
Using Bayes’ theorem, the likelihood that an item I_i is positive given the set of key presses C^w is:

$$P(I_i \mid C^w) = \frac{P(C^w \mid I_i)P(I_i)}{P(C^w)}$$

- $P(C^w) = \prod_k P(C_k^w)$ is the probability that keypresses exist, i.e. the probability that a set of items has keypresses from worker w.
  $\prod_k P(C_k^w) = P(C_1^w) \times P(C_2^w) \times \ldots . \times P(C_k^w) = P(C_k^w)^k = constant$, assuming that these probabilities are equal, i.e. it’s equally likely that a worker reacts to any item.
- $P(I_i)$ is the probability of an item being correctly identified in any case. Using the law of total probability, the probability of correctly identifying item I_i in any case is:

  $$\begin{aligned} P(I_i) &= P(I_i) \times 1 = P(I_i) \times [P(C^1) + P(C^2) + \cdots + P(C^w)] \\ &= P(I_i) \times P(C^1) + P(I_i) \times P(C^2) + \cdots + P(I_i) \times P(C^w) \\ &= \sum_{w=1}^{W} P(I_i \mid C^w)P(C^w) \end{aligned}$$

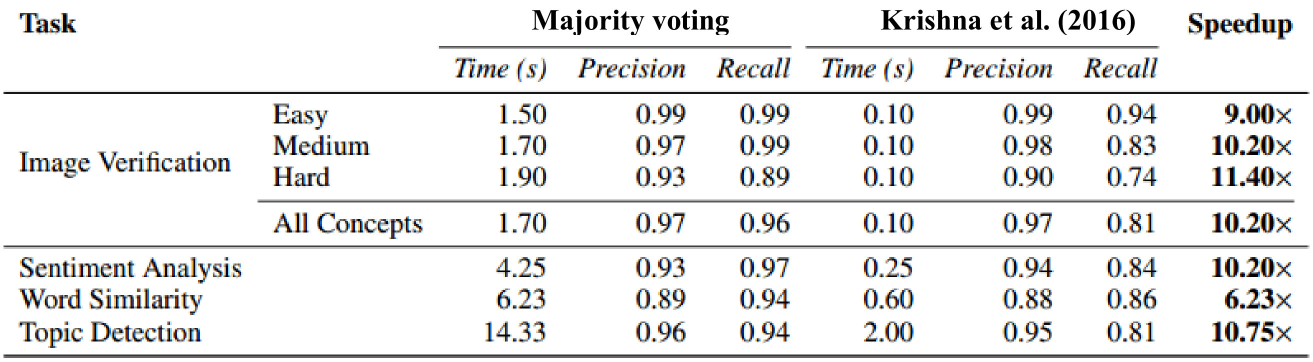
  $P(I_i)$ can be a constant, or it can be an estimate from a machine learning algorithm specific to a domain. For example, if the goal is to scale up a dataset of “dog” profiles, one could use a small set of known “dog” profiles to train a binary classifier and use that to calculate $P(I_i)$ for all the unknown profiles. For images, the authors use a pretrained convolutional neural network to extract image features and train a linear support vector machine to calculate $P(I_i)$.
- $P(C^w \mid I_i)$ is a set of independent keypresses for item I_i:

  $$P(C^w \mid I_i) = P(C_1^w, C_2^w, \ldots, C_k^w \mid I_i) = \prod_k P(C_k^w \mid I_i)$$

  Each keypress C_k^w is modeled as a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ given a positive item. The authors train the mean μ and variance σ by running rapid crowdsourcing on a small set of items for which the positive items are already known.

The model works by treating each keypress as creating a Gaussian “footprint” of probability of an image being correctly identified about 400ms before the keypress (Figure 4). The alignment of the peak of the Gaussian distribution under each ground truth image is arbitrary. Now that there are a set of probabilities for each item, the next step is to decide which ones should be classified as being correctly identified. Each set of items I is ordered according to likelihood of being in the correctly identified class $P(I_i)$. All items above a certain threshold are set as correctly identified. This threshold is a hyperparameter that can be tuned to trade off precision versus recall. In total, this model has two hyperparameters: (1) the threshold above which images are classified as correctly identified and (2) the speed at which items are displayed to the user. Both hyperparameters are modeled in each topic, for example, image verification or sentiment analysis. For a new topic, first, an estimate is obtained regarding how long it takes to label each item in the conventional setting with a small set of items. Next, the speed at which each item is displayed is reduced in portions until the model is unable to achieve the same precision as the untimed case.

To evaluate this method's performance, Krishna et al. (2016) examines its precision and recall values. Precision is the proportion of true positives among true and false positives. Recall is the proportion of true positives that are correctly identified. As an example, when labeling a dataset such as that of ten thousand articles about housing, precision is often the highest priority: articles labeled as on-topic by the system must in fact be about housing. Recall, on the other hand, is of lower priority, because there is often much unlabeled data: even if the system misses some on-topic articles, the system can label more items until it reaches the desired dataset size. Thus, the goal of this approach is to produce high precision at high speed, sacrificing some recall if necessary. Indeed, this approach achieves very high precision - the same as the approach of punishing all errors during annotation (Table 1). This approach also achieves the speed of about 6 to 11 orders of magnitude times more than the known approach.



Task		Majority voting			Krishna et al. (2016)			Speedup
		Time (s)	*Precision*	*Recall*	*Time (s)*	*Precision*	*Recall*	
Image Verification	Easy	1.50	0.99	0.99	0.10	0.99	0.94	**9.00×**
	Medium	1.70	0.97	0.99	0.10	0.98	0.83	**10.20×**
	Hard	1.90	0.93	0.89	0.10	0.90	0.74	**11.40×**
	All Concepts	1.70	0.97	0.96	0.10	0.97	0.81	**10.20×**
Sentiment Analysis		4.25	0.93	0.97	0.25	0.94	0.84	**10.20×**
Word Similarity		6.23	0.89	0.94	0.60	0.88	0.86	**6.23×**
Topic Detection		14.33	0.96	0.94	2.00	0.95	0.81	**10.75×**

Table 1. Image verification: "easy" – dogs, "medium" – person riding motorcycle, "hard" – eating breakfast. Sentiment analysis – find positive tweets, word similarity – find synonyms of a word, topic – match articles with topics, such as "housing" or "gas". Speedup column is written in terms of order of magnitude M in the form $M = 10^n$. E.g. 9.00× speedup means $M = 10^n = 9.00$, thus $n = \log_{10} 9.00 = 0.95$. Similarly, 10.20× speedup means $M = 10^n = 10.20$, thus $n = \log_{10} 10.20 = 1.009$. Subsequently, approximately an order of magnitude increase in speed was achieved (Krishna et al., 2016).

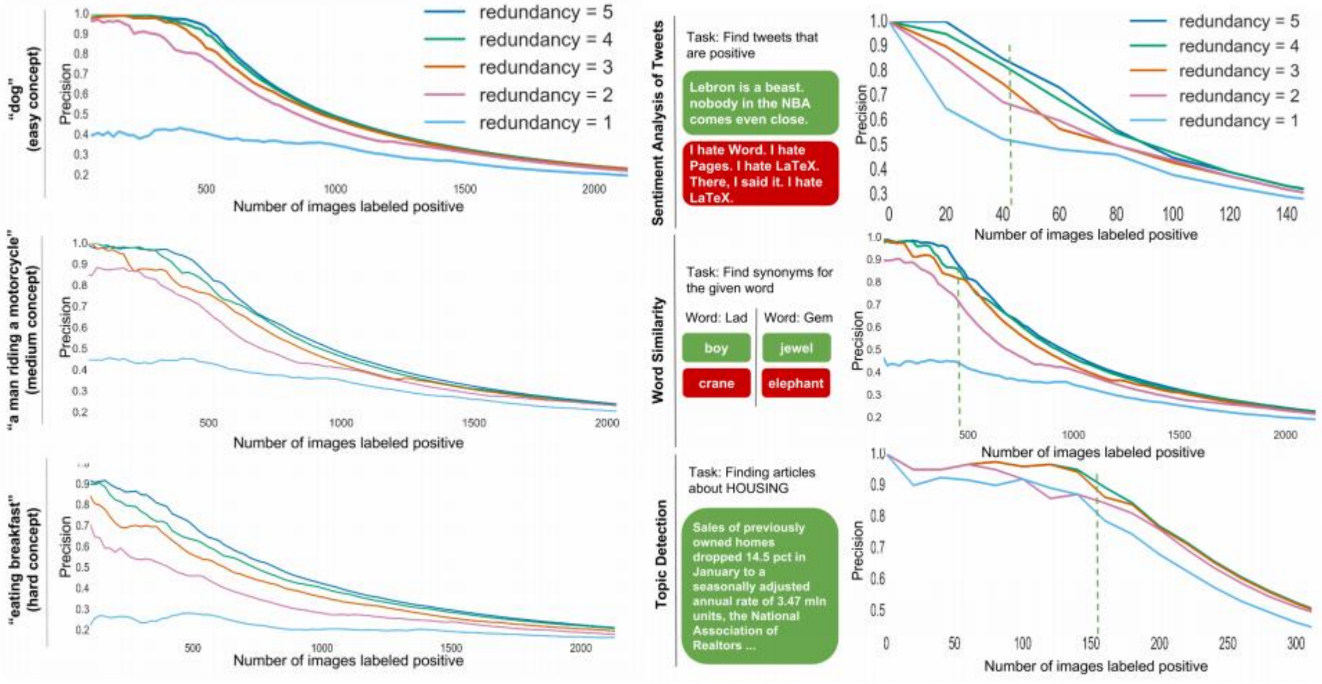


Figure 5. Precision of the method in binary classification tasks (Krishna et al., 2016).

The cognitive load on workers was measured using the NASA Task Load Index (TLX) [12] on all tasks, including image verification. TLX measures the perceived workload of a task. A survey was run on 100 workers who in majority voting approach and 100 workers who used Krishna et al. (2016) technique across all tasks. The cognitive workload for their method was slightly elevated but not statistically significant (61.1 vs. 70.0).

This method requires few algorithms yet achieves good results enough to be applied to many annotation projects for the purpose of increasing speed while still maintaining quality, as well as turning errors into correct labels. We can do so by collecting the amount of time a worker does each task, treating this duration as a Gaussian distribution, and training a model to predict the correct labels using this duration.

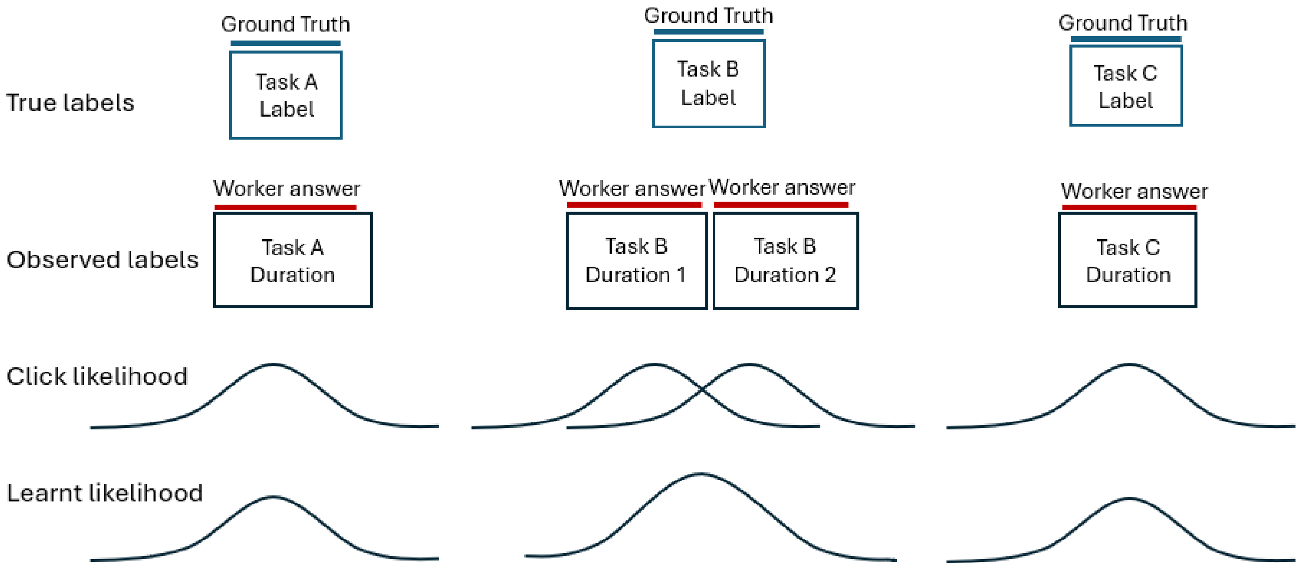


Figure 6. Applying Krishna et al. (2016) method to general data labeling tasks.

First, we need a subset of ground truth labels. These ground truth labels serve two purposes. The first purpose is to filter out low-quality work. Tasks with ground truth labels will be mixed with task of unknown labels. Workers who get zero of the correct labels over a period of time will be rejected. The second purpose is to train a model to calculate a duration time given a correct label. This value is the $P(I_i)$ value discussed above. The ground truth labels can be created and verified by annotators, admins, and reference materials.

Second, we need a subset of labels for tasks with unlimited timing. This subset is used to determine the lowest duration possible given a task. The workers are allowed to work without a time limit to determine an initial average duration. This duration is then successively reduced to until workers start making mistakes. This is the error point, which will later be used in the model.

Third, we set up the model. Here are the model details. A set of labels $L = \{L_1, \dots, L_n\}$ are sent to W workers. The labels in the set are in a different random order for each worker. From a worker w, a set of durations $D^w = \{d_1^w, d_2^w, \dots, d_k^w\}$ is collected, where $w \in W$ and k is the total number of keypresses from worker w. The aim is to calculate the probability of a label being correct given a set of durations, $P(L_i \mid D^w)$.

Using the law of total probability: if $D_1^w, D_2^w, \dots, D_k^w$ are partitions of the sample space S, then the probability of correctly identifying label L_i in any case is:

$$P(L_i) = P(L_i) \times 1 = P(L_i) \times [P(D_1^w) + P(D_2^w) + \cdots + P(D_k^w)] = \sum_w P(L_i \mid D^w) P(D^w)$$

According to Bayes' theorem, the likelihood of a label L_i being correct given the set of key durations D^w is:

$$P(L_i \mid D^w) = \frac{P(D^w \mid L_i\,)P(L_i)}{P(D^w)}$$

- $P(D^w) = \prod_k P(D_k^w)$ is the probability that durations exist, i.e. the probability that a set of labels has durations from worker w.

  $P(D^w) = P(D_1^w) \times P(D_2^w) \times \ldots . \times P(D_k^w) = P(D_k^w)^k = constant$, assuming that these probabilities are equal, i.e. it's equally likely that a worker reacts to any label.
- $P(L_i)$ is the probability of a label being correct in any case. $P(L_i)$ models the estimate of label l_i being correct. It can be a constant, or it can be an estimate from a machine learning algorithm specific to a domain. There are several options to calculate $P(I_i)$, such as using a linear support vector machine.
- $P(D^w \mid L_i\,)$ is a set of independent durations for label L_i:

  $$P(D^w \mid L_i\,) = P(d_1^w, d_2^w, \ldots, d_k^w \mid L_i) = \prod_k P(D_k^w \mid L_i)$$

  Each duration d_k^w is modeled as a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ given a correct label. We will train the mean μ and variance σ by using annotators on a small set of labels for which the correct labels are already known.

III. Achieve true human-like reasoning.

Large language models (LLMs) have demonstrated success in various tasks that require reasoning. However, evidence shows these models tend to rely on pattern matching and correlation rather than logical inference. For example, instead of solving a problem entirely by understanding core concepts, they find solutions that most likely resemble the correct solution based on their training.

Indeed, evaluations on multiplication and dynamic programming problems indicate that ChatGPT and GPT-4 struggle with multi-step reasoning tasks (Dziri et al., 2023). For instance, humans can solve 3-digit by 3-digit multiplication arithmetic after learning basic calculation rules. In the meantime, off-the-shelf ChatGPT and GPT4 respectively achieve only 55% and 59% accuracy on this task. Further examination shows that in multiplication and dynamic programming tasks, these models reduce multi-step reasoning computation graphs into subgraph pattern matching. (Dziri et al., 2023)

Wu et al. (2024) evaluated LLMs' ability to apply rules to novel scenarios. They did so by testing GPT-4, GPT-3.5, Claude, and PaLM-2 models on default tasks and counterfactual tasks. Counterfactual tasks are tasks in which specific input-output mapping functions are changed. In this study, a default arithmetic calculation is performed in base 10, while its counterfactual arithmetic calculation is performed in base 9 and other bases. For example, adding 27 and 62 results in 89 in base 10, and 100 in base 9. The expectation is as long as the model knows the rules of arithmetic and base, it shouldn't have a problem applying them. However, LLMs such as GPT 3.5, Claude and PaLM-2 struggle to arrive at the correct conclusion for two-digit additions in bases other than 10 (Figure 7).

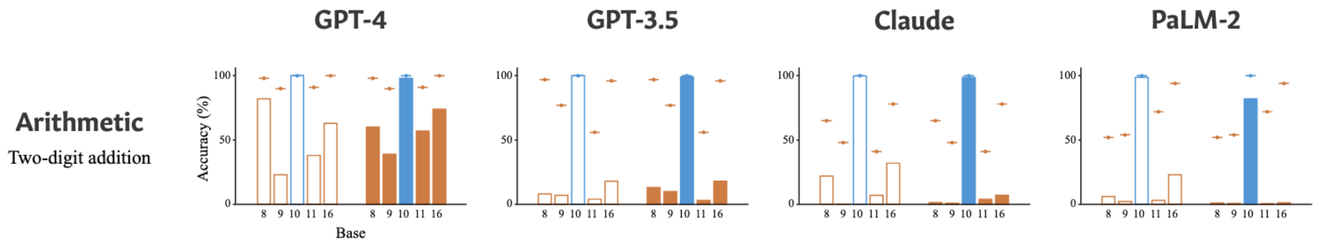


Figure 7. Performances of LLMs in two-digit additions in bases 8, 9, 10, 11, and 16. Blue bars represent default tasks; orange bars represent counterfactual tasks; unfilled and filled bars respectively represent including or not including the instruction "let's think step by step". (Wu et al., 2024)

With regards to spatial reasoning, LLMs also seem to struggle to apply the concept of orientation when being asked to identify the coordinates of objects in a 2D cardinal coordinate system. A default task is under the condition that east is in the direction of the x axis going from left to right. A counterfactual task is under the condition that east is in the direction of the x axis going from right to left. The accuracy of GPT-4 is below 25% when the east-west direction is swapped and below 15% when the axes are rotated or randomly permuted (Figure 8).

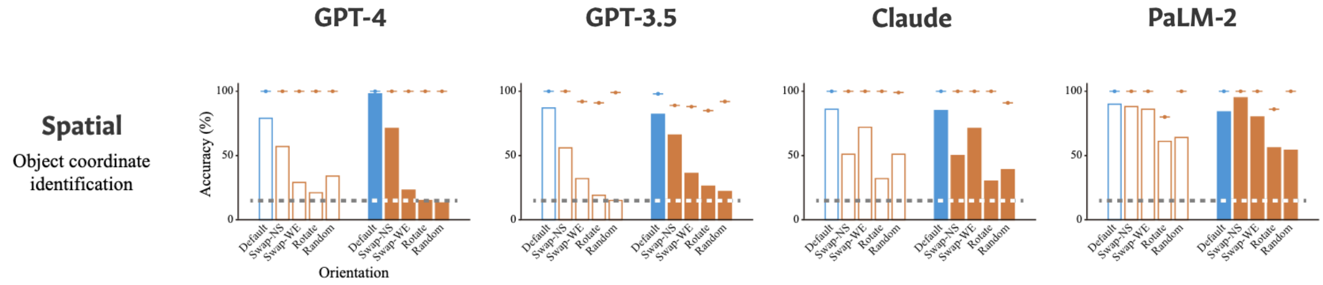


Figure 8. Performances of LLMs in identifying object coordinates when the north-south direction is swapped, east-west direction is swapped, or axes are rotated or randomly permuted. Blue bars represent default tasks; orange bars represent counterfactual tasks; unfilled and filled bars respectively represent including or not including the instruction "let's think step by step". (Wu et al., 2024)

LLMs are further asked to generate code that draws an object in the default task and generate code that draws the same object vertically flipped, rotated 90 degrees or rotated 180 degrees. The accuracy of GPT-4 is below 50% and the accuracy of GPT-3.5 and Claude is below 25%, while PaLM-2 often generates mal-formatted code.

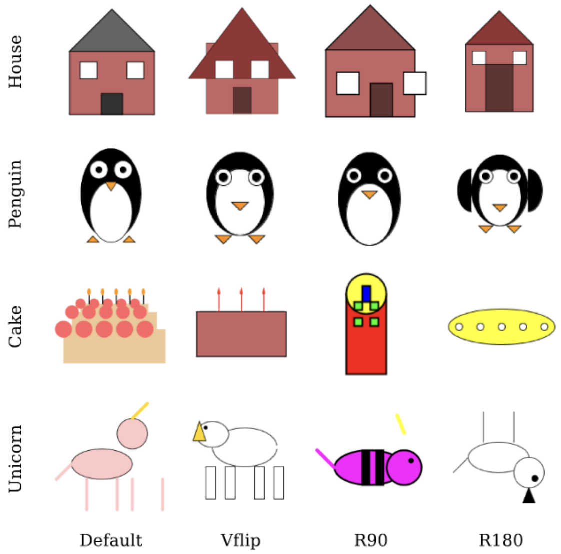
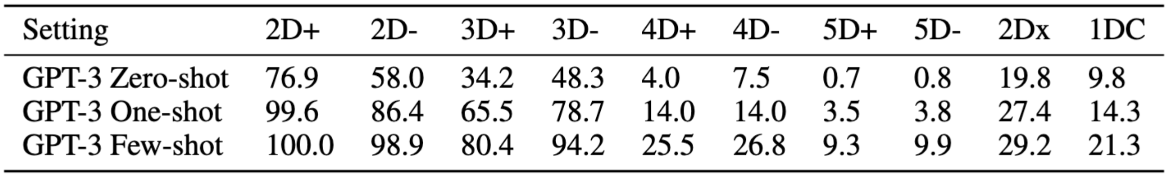


Figure 9. GPT-4 drawing visualizations under the default (upright) and counterfactual conditions: vertical flip (Vflip, i.e. upside-down), rotates 90 degrees (R90), and 180 degrees (R180). or the counterfactual settings, GPT-4 either does not transform the objects as instructed (e.g., house and penguin) or struggles to draw meaningful objects (e.g., cake and unicorn). (Wu et al., 2024).

Is this gap in the models' ability to apply rules to novel scenarios the result of lack of training? If the focus has been for the models to match patterns and recite correlations, it's understandable they are less likely to perform well in logical inference and application of rules to unfamiliar situations. Indeed, studies have shown LLMs achieve marked increase in accuracy when presented with *examples of reasoning* in the prompt.

Brown et al., (2020) uses GPT-3 with 175 billion parameters to demonstrate large language models are few-shot learners, i.e., learning from multiple examples. In zero-shot condition, the model is given only a natural language description of the task. For example, “translate English to French”, and the word “cheese”, and is expected to produce the translated word. In one-shot condition, in addition to the description, the model is given an example of the task, such as “sea otter -> loutre de mer”. In few-shot condition, the model sees a few more examples of the task. The result shows a marked increase in accuracy. For arithmetic tasks, the few-shot approach is more accurate than zero-shot approach, but a margin of 23.1% in two-digit addition, 30.9% in two-digit subtraction, and even 50.2% in three-digit addition (table 2). For word manipulation tasks, few-shot approach outperformed zero-shot approach by up to 59% (table 3).



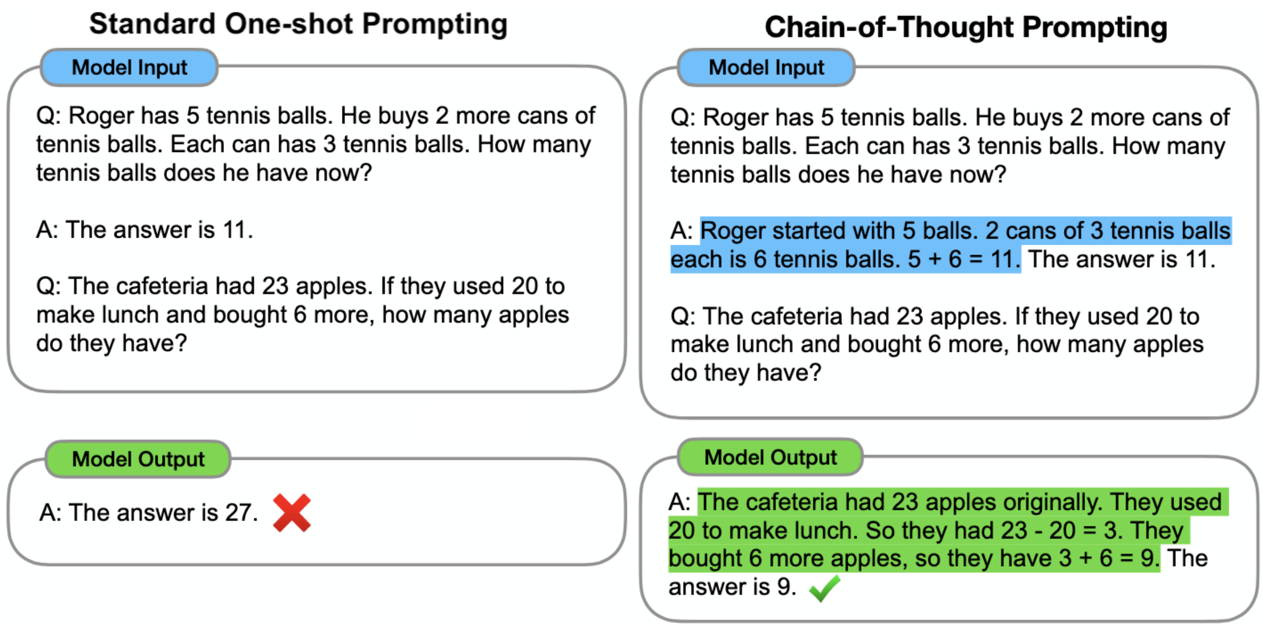
Setting	2D+	2D-	3D+	3D-	4D+	4D-	5D+	5D-	2Dx	1DC
GPT-3 Zero-shot	76.9	58.0	34.2	48.3	4.0	7.5	0.7	0.8	19.8	9.8
GPT-3 One-shot	99.6	86.4	65.5	78.7	14.0	14.0	3.5	3.8	27.4	14.3
GPT-3 Few-shot	100.0	98.9	80.4	94.2	25.5	26.8	9.3	9.9	29.2	21.3

Table 2. Results on basic arithmetic tasks for GPT-3 175B in zero-shot, one-shot and few-shot settings. {2,3,4,5}D{+,-} is 2-, 3-, 4-, and 5- digit addition or subtraction. 2Dx is 2-digit multiplication. 1DC is 1-digit composite operation. Results become progressively stronger moving from the zero-shot to one-shot to few-shot setting.

Setting	CL	A1	A2	RI	RW
GPT-3 Zero-shot	3.66	2.28	8.91	8.26	0.09
GPT-3 One-shot	21.7	8.62	25.9	45.4	0.48
GPT-3 Few-shot	37.9	15.1	39.7	67.2	0.44

Table 3. GPT-3 175B performance on various word unscrambling and word manipulation tasks, in zero-, one-, and few-shot settings. CL is "cycle letters in word", A1 is anagrams of but the first and last letters, A2 is anagrams of all but the first and last two letters, RI is "Random insertion in word", RW is "reversed words"

The few-shot approach is expanded to an approach that has shown repeated success in multiple studies called chain-of-thought reasoning (Wei et al, 2023). In addition to examples, this technique provides step-by-step reasoning in each example (Figure 10). As a result, the model learns ordered reasoning and applies the reasoning to its own answer as opposed to reciting a probable pattern shown in past training.



Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: The answer is 27. ✗

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Figure 10. Standard one-shot prompting provides an example, while chain-of-thought prompting provides a sample step-by-step reasoning for the correct output. Consequently, the model learns to apply this reasoning to produce the correct answer. (Wei et al., 2023)

Using this approach, models such as GPT-3 and PaLM have achieved a new state-of-the-art standard and even surpassed previous supervised best in solving moderate-difficulty math problems. At 175 billion parameters and 540 billion parameters, GPT-3 and PaLM both saw a marked increase of approximately 40% in solve rate (Figure 11).

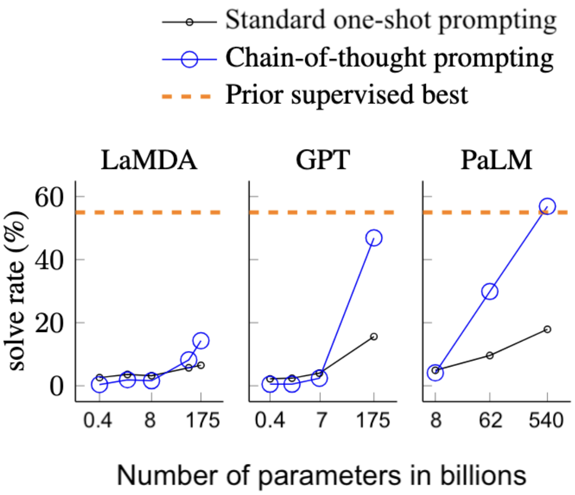


Figure 11. Chain-of-thought prompting performance in solving math problems using LaMDA, GPT-3 and PaLM models. The dataset used is GSM8K, a curated dataset of 8.5K grade school math questions and natural language solution of moderate difficulty.

With chain-of-thought prompting, sufficiently large models may be able to learn true reasoning with only a few instances of a task or even when seeing it defined only once (Brown et al., 2020; Wei et al., 2023). One caveat is this approach has only been shown to yield performance gains when used with models of ~100B parameters or more. Applying this approach in models of smaller scale, that is, of fewer number of parameters, produced fluent but illogical reasoning, leading to lower performance than few-shot prompting. An alternative approach for a smaller model is symbolic chain-of-thought distillation method (Li et al., 2024). First, multi-step rationales from a large model (such as GPT 175B) are sampled. Next, the small model (125M to 1.3B) is trained to predict the sampled rationales and sampled labels. Finally, human judges compare the chain-of-thought reasoning of the small model to the large model. This method has shown success in achieving chain-of-thought reasoning in small-scale models (Li et al., 2024).

In conclusion, data annotation process can be improved using various techniques as shown in the published literature. Selecting the best tool depends on the nature of the project and the purpose of the model. Some directions include adjusting the workflow such that there are multiple stages (Find-Fix-Verify) and breaking down larger tasks into smaller tasks (Partition-Map-Reduce). Gaussian distributions can be created for each worker keypress or task duration and then the keypress latency or task duration can be modeled to predict the correct labels from erroneous labels. Developing true logical inference in machines as opposed to pattern matching and route memorization may be accomplished using techniques such as chain-of-thought prompting, which trains models to produce step-by-step logical inference. Continuous study and development of the data labeling process can lead to a more streamlined workflow, higher quality at a lower speed, and improved machine's ability to achieve true understanding.

References

Bernstein, M., Little, G., Miller, R., Hartmann, B., Ackerman, M., Karger, D., Crowell, D., Panovich, K. (2010). Soylent: a word processor with a crowd inside. *UIST' 10*. https://up.csail.mit.edu/other-pubs/soylent.pdf

Branson, S., Wah, C., Schroff, F., Babenko, B., Welinder, P., Perona, P., Belongie, S. (2010). Visual recognition with humans in the loop. *Computer vision - ECCV 2010.* Springer. 438-451. https://doi.org/10.1007/978-3-642-15561-1_32

Wah, C., Branson, S., Perona, P., Belongie, S. (2011). Multiclass recognition and part localization with humans in the loop. *Computer vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2524-2541. https://doi.org/10.1109/ICCV.2011.6126539

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. *arXiv (Cornell University)*, *33*, 1877–1901. https://doi.org/10.48550/arxiv.2005.14165

Deng, J., Russakovsky, O., Krause, J., Bernstein, M., Berg, A., Fei-Fei, L. (2014). Scalable multi-label annotation. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3099-3102. https://doi.org/10.1145/2556288.2557011

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jian, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., Sanyal, S., Welleck, S., Ren, X., Ettinger, A., Harchaoui, Z., & Choi, Y. (2023). Faith and Fate: Limits of Transformers on Compositionality. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2305.18654

Ipeirotis, P., Provost, F., Wang, J. (2010). Quality management on Amazon Mechanical Turk. *Proceedings of the 14th ACM SIGKDD workshop on human computation.* ACM, 64-67. https://doi.org/10.1145/1837885.1837906

Jain, S., Graumann, K. (2013). Predicting sufficient annotation strength for interactive foreground segmentation. *Computer vision (ICCV), 2013 IEEE International Conference on*. IEEE, 1313-1320. https://doi.org/10.1109/ICCV.2013.166

Kittur, A. Smus, B., Khamkar, S., Kraut, R. (2011). CrowdForge: Crowdsourcing complex work. *UIST' 11*. https://static.googleusercontent.com/media/research.google.com/vi//pubs/archive/39980.pdf

Krishna, R., Hata, K., Chen, S., Kravitz, J., A Shamma, D., Fei-Fei, L., & S Bernstein, M. (2016). Embracing error to enable rapid crowdsourcing. *CHI 2016*. https://cs.stanford.edu/~kenjihata/papers/Embracing_CHI_2016.pdf

Li, L. H., Hessel, J., Yu, Y., Ren, X., Chang, K., & Choi, Y. (2023). Symbolic Chain-of-Thought distillation: small models can also “Think” Step-by-Step. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2306.14050

Sheng, V., Provost, F., Ipeirotis, P. (2008). Get another label? improving data quality and data mining using multiple, noisy labelers. *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 614-622. https://doi.org/10.1145/1401890.140196

Wah, C., Horn, G., Branson, S., Majie, S., Perona, P., Belongie, S. (2014). Similarity comparison for interactive fine-grained categorization. *Computer vision and pattern recognition (CVPR), 2014 IEEE International Conference on*. IEEE, 859-866. https://doi.org/10.1109/CVPR.2014.115

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. Le, Q., Zhou, D. (2023). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv (Cornell University)*. https://doi.org/10.48550/arXiv.2201.11903

Whitehill, J., Wu, T., Bergsma, J., Movellan, J., Ruvolo, P. (2009). Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. *Advances in neural information processing systems*. 2035-2043. https://dl.acm.org/doi/10.5555/2984093.2984321

Wu, Z., Qiu, L., Ross, A., Akyürek, E., Chen, B., Wang, B., Kim, N., Andreas, J., & Kim, Y. (2023). Reasoning or reciting? Exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2307.02477

Zhou, D., Basu, S., Mao, Y., Platt, J. (2012). Learning from the wisdom of crowds by minimax entropy. *Advances in Neural Information Processing Systems*. 2195-2203. https://dl.acm.org/doi/10.5555/2999325.2999380