Nemo: Guiding and Contextualizing Weak Supervision for Interactive Data Programming

Cheng-Yu Hsieh
University of Washington
cydhsieh@cs.washington.edu

Jieyu Zhang
University of Washington
jieyuz2@cs.washington.edu

Alexander Ratner
University of Washington
Snorkel AI, Inc.
ajratner@cs.washington.edu

ABSTRACT

Weak Supervision (WS) techniques allow users to efficiently create large training datasets by programmatically labeling data with heuristic sources of supervision. While the success of WS relies heavily on the provided labeling heuristics, the process of how these heuristics are created in practice has remained under-explored. In this work, we formalize the development process of labeling heuristics as an interactive procedure, built around the existing workflow where users draw ideas from a selected set of *development data* for designing the heuristic sources. With the formalism, shown in Figure 1, we study two core problems of (1) how to strategically select the development data to *guide* users in efficiently creating informative heuristics, and (2) how to exploit the information within the development process to *contextualize* and better learn from the resultant heuristics. Building upon two novel methodologies that effectively tackle the respective problems considered, we present Nemo, an end-to-end interactive system that improves the overall productivity of WS learning pipeline by an average 20% (and up to 47% in one task) compared to the prevailing WS approach.

PVLDB Reference Format:
Cheng-Yu Hsieh, Jieyu Zhang, and Alexander Ratner. Nemo: Guiding and Contextualizing Weak Supervision for Interactive Data Programming. PVLDB, 15(13): 4093 - 4105, 2022.
doi:10.14778/3565838.3565859

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ChengYuHsieh/Nemo.

1 INTRODUCTION

Manual labeling and curation of training datasets has increasingly become one of the major bottlenecks when deploying modern machine learning models in practice. In response, recent weak supervision approaches, wherein cheaper but noisier forms of labels are used, have received increasing research and industry attention. In one recent paradigm for weak supervision, data programming (DP) [29], users are able to quickly create and manage large training datasets by encoding domain knowledge and labeling heuristics into a set of *labeling functions* (LFs). Each of these functions, serving as a weak supervision source, programmatically annotates a subset of data points with possibly noisy weak labels. As the LFs may have varying accuracies, different LFs could suggest conflicting votes on certain data points. As a result, researchers have developed various modeling techniques to denoise and aggregate the weak labels, provided by different weak supervision sources, into probabilistic training labels that can be utilized to train downstream models [11, 29, 30]. Despite its recent emergence, the DP paradigm has powered many industrial-scale systems across various domains [4, 7, 28, 31].

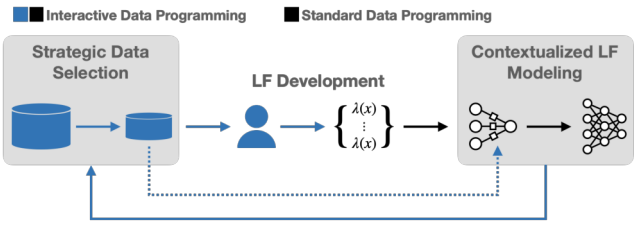

Figure 1: Interactive Data Programming (IDP) considers the entire data programming (DP) pipeline as an iterative cycle with two core problems of (1) strategic data selection and (2) contextualized LF modeling. The blue-colored components/steps in IDP are typically neglected in existing DP pipeline (the black-colored sub-procedures) focusing more exclusively on modeling and learning from a *given* set of LFs.

In leveraging the DP paradigm, the crucial—yet understudied—first step is to develop a set of LFs as the weak supervision sources. In practice, users typically look at a small set of data samples selected from the unlabeled dataset, called the *development data*, to draw ideas for writing LFs [28]. As a result of this common workflow, we especially note that the LFs developed are directly affected by and biased with respect to the data seen by the users during the development process. Particularly, an LF created with heuristics extracted from certain development data is more likely to generalize, or to provide labels, to those examples possessing similar patterns as the development data. In addition, among the covered examples, the LF may also be expected to perform more accurately on those examples within closer proximity to the development data, whereas having higher possibility of over-generalizing (to provide wrong labels) on the examples that lie in data subspaces further away from the development data. We visualize such trend in Figure 2.

Example 1.1. Consider a sentiment classification task on product reviews from various categories, as illustrated in Figure 3. First, by looking at reviews from a certain category, users would more likely create LFs that cover reviews from the same category. For instance, by looking at "Food" product reviews, potential LFs extracted such

Proceedings of the VLDB Endowment, Vol. 15, No. 13 ISSN 2150-8097.
doi:10.14778/3565838.3565859

as "delicious → positive" are more likely to generalize to other "Food" product reviews than to "Electronics" product reviews. Second, an LF created from a certain category may be expected to be more accurate for reviews in the same or similar categories. For instance, by looking at reviews from the "Movie" category, users may find "funny → positive" an useful LF. However, the LF appears less accurate for reviews from "Food" category, since "funny" could indicate negative sentiment when associated with taste/food.

While the data samples seen in the LF development process directly impact the resultant set of LFs created and provide valuable contextual information about LFs, existing work on DP have largely considered the entire LF development process as an exogenous black-box to the DP learning pipeline [11, 28–30]. The lack of attention on the LF development process thus poses three major limitations to the current DP paradigm:

- **Under-formalized LF Development Workflow:** The lack of formalism on the LF development process has obscured systematic study to optimize the workflow, making it less organized and more challenging for practitioners to design LFs for DP applications [6, 13, 38, 40].
- **Inefficient Development Data Selection:** Current LF development workflow selects development data with the most straightforward approach, uniform random sampling, which unfortunately can be time-consuming as it oftentimes requires users to inspect a considerable amount of data samples to create an informative set of LFs.
- **Dropped Data-to-LF Lineage:** The development context within which the LFs were developed is neglected, i.e., from which development data an LF was created, leaving behind valuable information about the LFs' expected accuracies in different data subspaces.

In this work, we make three corresponding hypotheses to tackle the limitations and improve the productivity of DP learning pipeline:

- **Formalism of LF development process could allow systematic study and optimization of DP workflow.** By framing the LF development process formally with clear objective, we would be able to study and improve the development process, making the DP paradigm more productive.
- **Strategic development data selection could lead to more efficient DP pipeline.** By strategically selecting the development data to be used by the users for writing LFs, instead of random sampling, we could ideally *guide* the users in efficiently creating a set of LFs that provide the most informative supervision signals for learning the subsequent models.
- **Exploitation of LF lineage to development data could enable more effective DP learning.** By tracking the LF lineages to the development data, we could ideally exploit this information to *contextualize* where the LFs are expected to perform best, and accordingly model the data-conditional accuracies of the LFs for more effective denoising.

Accordingly, we then make the following technical contributions that *positively* verify each of the above hypotheses:

Formalism of LF Development and First End-to-End System Solution: We formalize a broader DP paradigm, called Interactive Data Programming (IDP), that explicitly considers LF development as one central and interactive component in the entire learning workflow. As illustrated in Figure 1, we formulate the entire DP pipeline as an iterative human-in-the-loop cycle. In each iteration, the user creates new LFs based on the selected development data, and subsequently train the downstream model based on the LFs collected so far. The goal is to train an accurate model for the target task in as few iterations as possible. Towards the goal, we focus on studying two core problems: (1) how to intelligently select development data so as to guide users in developing most useful LFs efficiently, and (2) how to leverage the LF lineage to development data for facilitating more effective modeling and learning from the created LFs.

With the problem formalized, we then set out to design *Nemo*, the first end-to-end system for IDP which is built on top of two novel methodologies that respectively tackle the data selection and contextualized LF modeling problems. Through extensive quantitative evaluations and a carefully conducted user study, we validate our first hypothesis where we observe that Nemo offers significant performance lift over the current prevailing DP system an average 20%. When compared to other interactive learning schemes such as traditional active learning, Nemo as well leads to an average 34% performance lift.

Intelligent Development Data Selection Strategy: In building Nemo, we propose a novel data selection strategy, *Select by Expected*

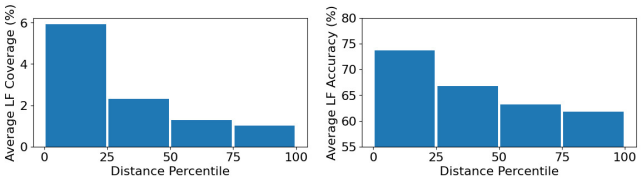

Figure 2: LFs generally have higher coverage (left) and accuracy (right) on the data subspace within closer proximity to their development data. For each LF, we organize all examples into 4 subspaces based on the percentile of their distance to the development data, and compute the LF's coverage/accuracy in each data subspaces. The plots are averaged results over 100 LFs on Amazon Review Dataset.

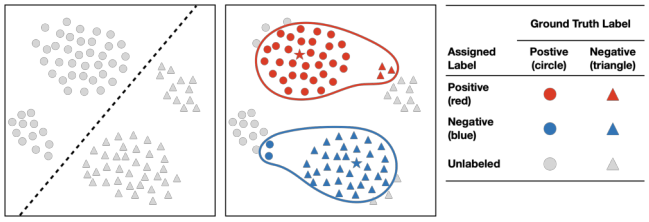

Figure 3: Left: A toy sentiment classification dataset with 4 clusters, each corresponding to product reviews from a category. Right: Looking at development data points (stars), users are likely to create LFs that generalize to similar examples. In addition, the LFs may be expected to be more accurate around the development data. Circle/Triangle corresponds to "ground truth" Positive/Negative label. Red/Blue corresponds to "assigned" Positive/Negative label and Gray is unlabeled.

Utility (SEU), to efficiently guide users in developing useful LFs. The key idea within the proposed SEU approach is to first *statistically measure the utilities* of the potentially generated LFs, and in turn select the examples that are expected to lead to high utility (or more informative) LFs, in which the expectation is calculated through a proposed *user model* that measures the conditional probability of user returning an LF by looking a specific development data (illustrated in the development data selector in Figure 4). With SEU, we validate our second hypothesis and improve the DP performance an average 16% compared to random sampling baseline.

Contextualized LF Modeling with Data Lineage: In Nemo, we propose a model-agnostic method that exploits the LF lineage to the development data to more effectively denoise and learn from the LFs. Particularly, based on the natural tendency for users to create LFs that are more precise around the neighborhood of the development data [3], we propose to refine each LF to be active only within a certain radius from their corresponding development data points (illustrated in LF contextualizer in Figure 4). With the method, we validate our third hypothesis and improve the DP performance an average 11% compared to the standard learning pipeline without leveraging the LF development context in modeling the LFs.

2 PRELIMINARIES

We review the standard data programming (DP) setup [28, 29]. Let each example $x \in \mathcal{X}$ be associated with a corresponding label $y \in \mathcal{Y}$, where the joint density is governed by some underlying distribution $\mathcal{D}$. Given a set of n unlabeled examples $U = \{x_i\}_{i=1}^n$, drawn from distribution $\mathcal{D}$, with their labels $\{y_i\}_{i=1}^n$ unobserved, the standard DP pipeline follows three main stages:

(1) **Labeling Function Development Stage:** Users encode labeling heuristics into a set of m labeling functions (LFs), $\{\lambda_j\}_{j=1}^m$, where $\lambda_j : \mathcal{X} \rightarrow \mathcal{Y} \cup \{0\}$. Each LF λ_j could either provide an example x_i with a possibly noisy label $\lambda_j(x_i) \in \mathcal{Y}$ or abstain with $\lambda_j(x_i) = 0$.

(2) **Label Denoising/Aggregation Stage:** The provided LFs are individually applied to the unlabeled examples, generating a label matrix $L \in \mathbb{R}^{n \times m}$ where $L_{ij} = \lambda_j(x_i)$. Then, a *label model* is learned to aggregate L, the votes on the training labels, into a set of probabilistic estimates of the ground truth labels $\{P(y_i|L)\}_{i=1}^n$.

(3) **End Model Learning Stage:** Finally, these probabilistic estimates could serve as probabilistic soft labels that can be used to train a final *discriminative model* $f : \mathcal{X} \rightarrow \mathcal{Y}$ that could generalize and make predictions on unseen examples, with the overarching goal of minimizing the generalization error on distribution $\mathcal{D}$.

In the LF development stage, users generally refer to some *development data*, sampled from the unlabeled set U, to extract labeling heuristics for writing LFs. As a result of this workflow, the data seen in the development stage has a direct influence on what LFs would be created, and on which examples these LFs may be expected to perform best (see Figure 2 and Example 1.1).

Despite the impact that development data have on the resultant LFs, the problems of how to strategically select these data for guiding LF development and how to potentially exploit this lineage in modeling the LFs have remained under-explored. In fact, the entire LF development process has been given scant attention by literature to date, where the most straightforward but naive *random sampling* has been the dominating approach for selecting the development data, and the modeling of LFs has been blind to their development context [11, 25, 28–30]. Unfortunately, as discussed in previous section, this current workflow renders the entire DP pipeline less efficient and effective.

3 INTERACTIVE DATA PROGRAMMING

To study the impact of LF development process in the DP paradigm, we formalize a broader DP framework that considers LF development as one central, iterative process within the entire DP learning pipeline. We term this new formalism Interactive Data Programming (IDP), as illustrated in Figure 1. We highlight two novelties in IDP compared to the standard DP paradigm:

- First, we formalize the LF development stage as a two-step process where (1) a subset of development data would first be selected from the unlabeled set, and (2) the users then develop LFs based on these selected data. While this is an established workflow in practice, this process has not been carefully formalized and studied in the literature.
- Second, we consider the LF development stage and the subsequent label/end model learning stage as interleaved steps in an interactive cycle, rather than a sequential and independent procedures. In each iteration, the users develop new LFs *guided* by the learning models, and the models in turn learn from the LFs created where the models are *aware* of the context within which the LFs are developed.

The overarching goal of IDP is to achieve highest end model predictive performance, evaluated with a held-out *test dataset*, in as few interactive iterations as possible. This entails that the users could efficiently design useful weak supervision sources, the LFs, and effectively train an accurate predictive model from the LFs. Specifically, we study two main problems towards the goal:

- First, we consider the problem of how to intelligently *select* examples to *efficiently* guide users in writing informative LFs from which an accurate predicitve model can be learned.
- Second, we consider the problem of how to exploit the *LF development context* to *effectively* model and learn from a given set of LFs.

Connection to Active Learning. We note that the first problem studied in IDP is analogous to active learning [34], in the sense that the goal is to iteratively select data points from an unlabeled dataset and solicit supervision feedbacks from the user to train an accurate downstream model, with as few queries to the user as possible. However, unlike active learning where in each iteration the user provides supervision in terms of a *single label annotation* to the selected data point, the supervision form provided in IDP is at a higher functional level (i.e., LFs) that noisily *annotates multiple data points at a time*. We emphasize that the fundamental difference between label annotations and LFs leads to a unique set of challenges in designing the data selection algorithm for IDP. In addition, the inherent noise comes with the LFs in IDP creates a unique second problem of the need to contextualize the expected accuracy of an LF on different data subspaces, which marks another major difference to active learning.

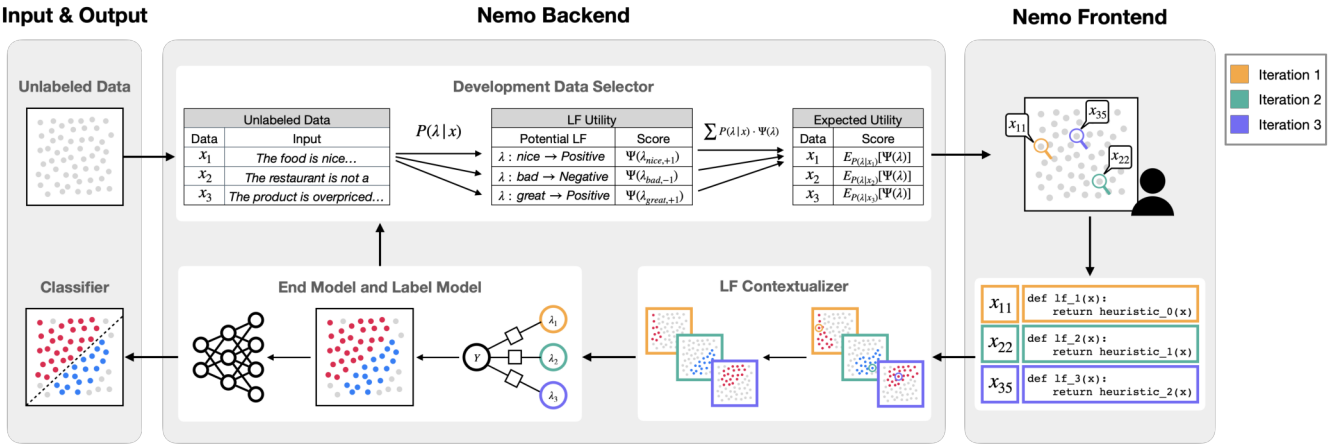

Figure 4: Nemo system overview. In each iteration, (1) the data selector intelligently picks an example from the unlabeled set based on current labeling information. (2) The user creates an LF based on the selected example. (3) The LF contextualizer refines each LF based on their development context, after which the label and end model is learned from the refined LFs.

A Subsuming Framework. IDP is an encompassing framework that subsumes many existing usages of DP in practice [3, 9, 28, 29]. For example, the current widely adopted DP workflow corresponds the vanilla instantiation of IDP that selects development data with random sampling and models the LFs without considering their development context [28]; the rule-exemplar learning approach [3] is another instance under IDP that supports half of the IDP loop where it models the LFs with their development context using the ImplyLoss model [3], but does not consider strategically selecting the development data.

Setup. Formally, given an unlabaled dataset $U = \{x_i\}_{i=1}^n$, the proposed IDP framework proceeds iteratively by the following steps:

(1) **Development Data Selection Stage:** In the t-th iteration, a subset of examples $S_t \subset U$ are strategically selected from the unlabeled set and shown to the user to guide development of LFs that are most informative to the models.

(2) **Labeling Function Development Stage:** Based on S_t, the user writes a set of k labeling functions $\Lambda_t = \{\lambda_{t1}, \ldots, \lambda_{tk}\}$ which extracts meaningful labeling heuristics encoded in the examples. The lineage of these LFs to the development data S_t is tracked and represented as a tuple (Λ_t, S_t).

(3) **Label/End Model Learning Stage:** Given the set of LFs created so far with their data lineage, $\{(\Lambda_1, S_1), \ldots, (\Lambda_t, S_t)\}$, the label and end models are learned from the LFs as in the standard DP pipeline, but with additional access to the LFs' development context. Finally, the current model information would be passed back to the data selection stage to start the next cycle; or the iteration stops and the end model is output for the learning task of interest.

Paper Scope. In the remainder of this paper, we focus on binary classification tasks where $\mathcal{Y} = \{-1, 1\}$ for ease of exposition. In addition, we focus on an atomic IDP setting where in each iteration, a single example is selected as the development data ($|S_t| = 1$), and the user in return provides a single LF ($|\Lambda_t| = 1$). Despite its seemingly simplification, we emphasize that after multiple interactive iterations, the user would have seen multiple development data points, and would have developed multiple LFs out from the data points. As a result, we can see how this atomic setup effectively builds up to the general IDP setup where the user may develop multiple LFs from multiple examples in each iteration. A subtle difference lies in that, in the atomic one-example to one-LF setup, the user looks at the data points and develops LFs *sequentially* rather than *in batch*. This sequential nature allows most efficient use of the user's effort as the underlying development data selection algorithm can adjust dynamically *given every newly developed LF*, avoiding the user spending extra effort in designing redundant LFs. While we focus on the atomic IDP setting in the paper, we provide discussions on how our proposed solutions can be generalized to support the general IDP setup.

Finally, we note the development context of Λ_t includes all previous sequence of development data the user has seen $(S_1, \ldots, S_t)$. In this work, we only consider the context window to include the data the user is currently looking at (i.e., S_t), and leave the incorporation of longer weighted context-sequence as a future direction.

4 NEMO ARCHITECTURE

We present Nemo, the first end-to-end system designed to support the full IDP loop by tackling the two core IDP problems. We provide Nemo system overview in Figure 4. Nemo consists of an user-facing frontend where users develop LFs based on selected development data and a suite of backend engines where the system computes the best example to select in each iteration and learns from the created LFs along with their development context. At the frontend, Nemo provides convenient user interface for users to easily create LFs based on selected development data (Section 4.1). In the system backend, the *Development Data Selector* strategically selects examples from the large unlabeled set to guide users in creating informative LFs (Section 4.2). In addition, the *Labeling Function Contextualizer* exploits the LF lineage to development data to facilitate more effective modeling and denoising of the LFs (Section 4.3).

Nemo workflow begins by initially taking as input an unlabeled dataset. Then, Nemo proceeds in an interactive loop where each iteration follows the three main IDP stages as illustrated in Figure 4. We provide the system pseudo-code in Appendix A.

System Configuration and Inputs. Nemo focuses on one most widely adopted type of LFs, the *primitive-based LFs*. Formally, we consider primitive-based LFs to be any function $\lambda : \mathcal{X} \rightarrow \mathcal{Y}$ that can be expressed by:

$$\lambda_{z,y}(x) : \texttt{return } y \texttt{ if } x \texttt{ contains } z \texttt{ else abstain}$$

where $z \in \mathcal{Z}$ is some domain-specific primitive and $y \in \mathcal{Y}$ is a target label. Such functional type (or family) of LFs has been widely considered in the literature [3, 6, 37, 41], and can flexibly capture arbitrary input pattern by defining the primitive domain $\mathcal{Z}$ accordingly. One representative instantiation in text domain is the keyword-based LFs, where $\mathcal{Z}$ is a set of keywords, e.g., n-grams. We note that the primitive-based LF form absorbs any uni-polar LF [30] that can be expressed as $\lambda : \mathcal{X} \rightarrow \{y, 0\}$, where $y \in \mathcal{Y}$. This is because we can effectively express any LF of form $\lambda : \mathcal{X} \rightarrow \{y, 0\}$ as "$\lambda'_{z_\lambda, y}(x) : \texttt{return } y \texttt{ if } x \texttt{ contains } z_\lambda \texttt{ else abstain}$", where the *contain* operator can be formally expressed as $z_\lambda = \mathbb{1}_{\lambda(x)=y}$. The key idea is that the primitive domain can be arbitrary defined by the user to contain any black-box transformation on the data. Finally, we note that uni-polar LFs, that map an input to a single class or abstain, are arguably the most common type of LFs used in practice [30].

Before the interactive loop starts, we ask the user to configure Nemo by specifying : (1) the data domain $\mathcal{X}$, (2) the label space $\mathcal{Y}$, and (3) the primitive domain $\mathcal{Z}$. These specifications allow Nemo to configure the user interface accordingly and later help users more conveniently develop LFs, as we shall illustrate shortly in Figure 5.

Example 4.1. Consider the running example of sentiment classification on product reviews. The data domain $\mathcal{X}$ is the text inputs. The label space is $\mathcal{Y} = \{positive, negative\}$. The primitive domain $\mathcal{Z}$ could be specified as the set of all uni-grams contained in the unlabeled set, which can be automatically inferred by Nemo given U.

4.1 User Interface: Writing Labeling Functions with Development Data

We describe how we design Nemo user interface to allow users conveniently create LFs guided by selected development data.

Creating LFs with Development Data. In DP applications, we observe a common practice that users would go through when developing LFs based on development data. Specifically, by looking at a data sample x, users generally follow three principal steps to create an LF:

(1) Determine the corresponding label y of the example x.
(2) Look for a primitive z within x that is indicative of the label y, which is expected to generalize well to other examples.
(3) Create and return the labeling function $\lambda_{z,y}$.

We note that such procedure has been widely adopted in practice and in previous literature [3, 9, 28]. The Nemo user interface is designed to support this workflow where users could easily create LFs from development data through a few mouse-clicks. We demonstrate the Nemo user interface using the running example on the sentiment classification task in Example 4.2.

Example 4.2. Figure 5 shows the Nemo user interface. In each iteration, the user will be shown a development data selected from the unlabeled set. In this case, a product review: "Perfect for my workouts...". To extract useful heuristic as LF from the example, the user would first determine that the review corresponds to positive sentiment. Then, the user looks for a keyword primitive in the review that supports positive sentiment, e.g., the word "perfect". Finally, by simply clicking on the corresponding sentiment and keyword, Nemo automatically creates an LF $\lambda_{perfect,\ positive}$.

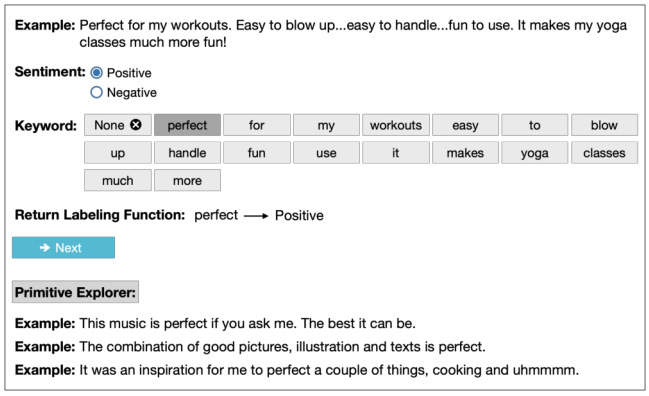

Figure 5: Nemo user interface. Users can easily create an LF from a development data by selecting a target label and a corresponding primitive. Note that in the example, the target labels—positive and negative—are configured according to the user provided label domain $\mathcal{Y}$. The shown candidate primitives are selected according to provided primitive domain $\mathcal{Z}$, in this case, the uni-grams in the dataset.

4.2 Development Data Selector: Guiding Informative Labeling Function Creation

The Development Data Selector is the core engine in Nemo that tackles the problem of how to guide efficient LF development through intelligently selected development data.

Random Selection Baseline. One straightforward selection approach is to *randomly* sample the development data from the unlabeled set in each iteration. In fact, as the development data selection problem has not been carefully considered in literature to date, the random selection baseline has been the prevailing approach adopted in most existing DP applications. However, a major drawback of the random approach is that it completely ignores the information provided by the current set of LFs on hand, potentially leading to the development of redundant LFs that provide little extra label information.

Example 4.3. In Figure 6, suppose that the user already developed two LFs, λ_1 and λ_2, for an initially unlabeled dataset, and we are already confident about the labels in two major clusters. Ideally, we would like the user to write LFs that provide supervision over the examples that have yet received label annotations. However, by random sampling, the selected development data has high probability of being an example within the two major data clusters, given the dominating probability mass of the two major clusters. In turn, the user will likely to create an LF that annotates the examples within the two clusters, providing limited extra supervision information and

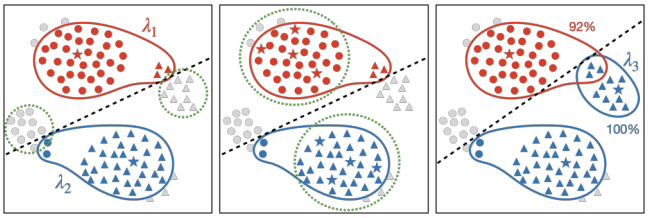

Figure 6: Left: Suppose we already have two LFs on hand. Middle: We see that random sampling may fail to solicit informative LFs from the user, where it has higher probability of selecting data points within the large, labeled clusters (the star points). Right: SEU selects the data point in the smaller unlabeled cluster as it has higher probability of leading to LFs that provide new, complementing label information.

starving the examples in the other two smaller clusters which as well play important roles in the classifier's decision boundary.

Ideally, in each iteration, we seek an LF that could best complement the current set of LFs, such that newly acquired LF could, for example, help provide coverage over an originally unlabeled data subspace, or help resolve the conflicts between current LFs. Thus, we call for an intelligent strategy that could more adaptively guide users in creating *useful* LFs according to the information on hand.

Select by Expected Utility. In designing the Nemo Development Data Selector, we propose a novel selection strategy, *Select by Expected Utility* (SEU), which adaptively picks the development data that *in expectation*, could guide the user in developing the *most informative* LF to complement the current collected label information. Formally, in each iteration t, SEU selects the development data by:

$$x^* = \arg\max_{x \in U} \mathbb{E}_{P(\lambda|x)}[\Psi_t(\lambda)], \tag{1}$$

where $P(\lambda|x)$ is the estimated conditional probability that an user would create LF λ given the development data being x, and $\Psi_t(\cdot)$ is an utility function measuring the informativeness of an LF λ based on the supervision provided by existing set of LFs Λ_t. SEU is shaped by two key properties in LF development:

- Development Tendency: By looking at different development data, the user is likely to create different LFs.
- Varying LF Utilities: Different resultant LFs provide different levels of useful supervision information.

Let $\mathcal{F} = \{\lambda_{z,y} | z \in \mathcal{Z}, y \in \mathcal{Y}\}$ be the LF family where all possible LFs lie. Given a development example x, SEU first leverages $P(\lambda|x)$, which we call the *user model*, to estimate the probability for each LF $\lambda \in \mathcal{F}$ to be created by the user. Then, SEU evaluates the informativeness of each LF $\lambda \in \mathcal{F}$ by the utility function $\Psi(\cdot)$. Finally, an example x's ability to lead to an informative LF λ can be summarized by the expected value of $\Psi(\lambda)$ taken with respect to $P(\lambda|x)$. We refer the readers to the Development Data Selector in Figure 4 for an illustration of the SEU selection strategy. By design, SEU achieves the goal of *selecting an example that is expected to lead to useful LFs* by capturing and leveraging the above two key properties in LF development via the user model and utility function respectively. We describe below how we design the user model $P(\lambda|x)$ and the utility function $\Psi(\cdot)$.

User Model. The goal of user model $P(\lambda|x)$ is to provide probability estimate for the user returning an LF λ given a development example x. We model this conditional probability for any LF $\lambda_{z,y} \in \mathcal{F}$ by:

$$P(\lambda_{z,y}|x) = \begin{cases} P(y) \cdot \frac{acc(\lambda_{z,y})}{\sum_{\lambda \in \{\lambda_{z,y} | z \text{ in } x\}} acc(\lambda)}, & \text{if } z \text{ contained in } x \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $acc(\lambda) = \frac{\sum_{\lambda(x_i) \neq 0} \mathbb{1}_{\{\lambda(x_i) = \hat{y}_i\}}}{\sum_{\lambda(x_i) \neq 0} 1}$ denotes the approximated accuracy of λ. The user model closely reflects the procedure of how users develop LFs in practice. Recall from Section 4.1, when given a data sample x, the user would (1) determine the corresponding label y for x, and (2) select a y-indicative primitive z from the candidate primitives contained in x to create the LF $\lambda_{z,y}$, which is expected to generalize well to other data points. By mirroring the procedure in parallel, the user model leverages chain rule to decompose $P(\lambda_{z,y}|x)$ into (1) the probability of y being the underlying label, and (2) the probability that z would be picked by the user from all candidate primitives $\{z \in \mathcal{Z} | z \text{ contained in } x\}$. In the user model, we utilize the label prior $P(y)$ to model the probability of y being the ground truth label for an example x. Then, we model the probability that z being picked to be proportional to how strongly z is indicative of the target label y, captured by the accuracy of $\lambda_{z,y}$. In the absence of the ground truth labels, we use the current label predictions from the discriminative model $\hat{y} = f(x)$ to approximately compute the true accuracies of the LFs. Finally, we note that the user model simply assigns zero probability to LFs that operate on primitives not residing in x, since these primitives would not be selected by the user from x.

LF Utility Function. The goal of LF utility function $\Psi : \mathcal{F} \to \mathbb{R}$ is to measure the informativeness of an LF, given the current collected supervision information. We design the LF utility function to be:

$$\Psi_t(\lambda) = \sum_{i \in C} \psi_t^{\text{uncertainty}}(x_i) \cdot (\lambda(x_i)\hat{y}_i), \tag{3}$$

where $C = \{i | x_i \in U, \lambda(x_i) \neq 0\}$ is the set of indices of those examples covered by λ, and $\psi_t^{\text{uncertainty}}(x_i) = -\sum_{y_i \in \mathcal{Y}} P(y_i|\Lambda_t) \cdot \log P(y_i|\Lambda_t)$ is the current *label model uncertainty* on an example x_i based on existing LFs Λ_t. The utility function $\Psi(\cdot)$ gives higher scores to LFs that provide *accurate* label annotations to the examples with *high label uncertainty*. Specifically, the utility score of an LF λ is the sum over the label uncertainty scores $\psi_t^{\text{uncertainty}}(x_i)$ of the examples to which λ provides labels, weighted by whether the provided label $\lambda(x_i)$ is correct or not, i.e., $\lambda(x_i)\hat{y}_i \in \{-1, 1\}$, where $\hat{y}_i$ is an approximate to the ground truth label. In DP, a high label model uncertainty score $\psi_t^{\text{uncertainty}}(x_i)$ corresponds to either an example that has not been covered by any LFs, or an example on which the LFs disagree the most. New label information on these uncertain data points provides informative supervision signals that reduce the label model's uncertainty over the entire dataset, complementing the existing LFs. *Correct* labeling to these uncertain data points allows us to obtain a more holistic view on the entire data space or help reduce the noise within the labeled data, both leading to *positive* influences to the model performances. On the other hand, *incorrect* labeling on these uncertain data points introduces influential noise into the dataset, resulting in *negative* impact that is likely to undermine the subsequent model performances. As a result, we

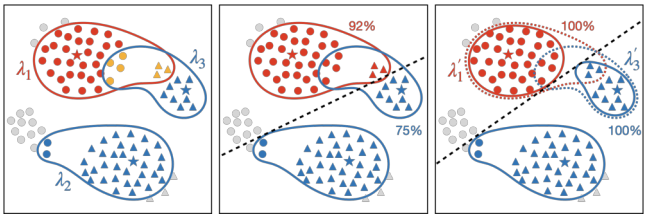

Figure 7: Left: λ_1 and λ_3 have conflicts on the yellow-colored points. Middle: In the ideal case where we could perfectly estimate the source accuracies, standard learning pipeline still fails to resolve the conflicts where it assigns wrong labels to the triangular points. Right: Contextualized learning pipeline refines the LFs and resolve the conflicts perfectly.

design $\Psi(\cdot)$ to take into account of both the informativeness and the accuracy of the supervision an LF provides for evaluating its usefulness to the DP learning pipeline. We provide the full SEU pseudo-code in Appendix A.

Example 4.4. In Figure 6, unlike random sampling, SEU aims to select development data based on their probability of leading to useful LFs. In this case, SEU would assign higher scores to data points in the smaller, unlabeled clusters (green-dash circled area) as they have higher probability of leading to LFs that provides new supervision signals complementing current labeling information.

4.3 Labeling Function Contextualizer: Modeling LFs with Development Context

The LF contextualizer is the main component in Nemo that tackles the problem of exploiting LF development context for more effective denoising and learning from the LFs.

Standard Learning Pipeline. In the DP learning pipeline without LF contextualizer, the standard procedure to learn from a given set of m LFs $\{\lambda_1, \ldots, \lambda_m\}$ is to first apply each LF to the examples $\{x_i\}_{i=1}^n$ to obtain a label matrix L where $L_{ij} = \lambda_j(x_i)$. Then, from the label matrix L, a label model is learned to estimate the accuracies of these LFs [11, 28–30]. The estimated accuracies are used as corresponding weighting terms in aggregating the votes provided by each LF to produce the probabilistic soft labels $\{P(y_i|L)\}_{i=1}^n$. The more accurate an LF is, the larger the weight its vote receives in the aggregation process. Notably, in most of the prevailing label model approaches [28–30], each LF is assumed to be uniformly accurate over all the data points it covers, i.e., each LF is modeled to have the same accuracy across the entire data space. Nonetheless, this modeling assumption is often violated in practice. Specifically, when an LF is developed by an user looking at specific development data, we observe that the LF is likely to be more accurate on the examples within closer proximity to the development data while having lower accuracy on examples that lie further away.

Example 4.5. Continuing from Example 4.4, consider a case that the newly returned LF, λ_3 in Figure 7, is not perfectly accurate, and there are conflicts (yellow-colored) between λ_1 and λ_3 . Even if we could perfectly estimate the source accuracy of both λ_1 and λ_3, we would still assign wrong labels to the triangular points, since λ_1 would receive higher weight over λ_3 for λ_1 having a higher overall accuracy.

Contextualized Learning Pipeline. Given the observation that LFs tend to provide noisier labels on examples that lie further away from their development data, we would ideally exploit this lineage to more effectively denoise and learn from the LFs. To this end, we propose a contextualized learning pipeline in Nemo where we leverage the LF contextualizer to refine and denoise the LFs—according to their development context—before feeding them as input weak supervision sources to learning the subsequent label and end discriminative model, as shown in bottom pipeline in Figure 4. In the Nemo learning pipeline, the LF contextualizer refines each LF by restricting it to be *active* only on examples within a certain proximity of its corresponding development data point, dropping the labels assigned to examples that are further away as these labels are prone to be noisier. Formally, let x_λ denote the development data point from which the LF λ is created. Given a set of m LFs and their corresponding development data points $\{(\lambda_1, x_{\lambda_1}), \ldots, (\lambda_m, x_{\lambda_m})\}$, the LF contextualizer refines each LF $\lambda_j \in \{\lambda_j\}_{j=1}^m$ into $\lambda_j^{'}$ by:

$$\lambda_j^{'}(x) = \begin{cases} \lambda_j(x), & \text{if } dist(x, x_{\lambda_j}) \leq r_j \\ 0 \text{ (abstain)}, & \text{otherwise} \end{cases} \tag{4}$$

where $dist(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ measures the distance (dissimilarity) between two input examples, and r_j is a given thresholding value for the refinement radius. By discarding the assigned labels to examples where the LFs are expected to perform more poorly, we may reduce the noise in the generated label matrix L, and in turn produce more accurate probabilistic soft labels $\{P(y_i|L)\}_{i=1}^n$. We provide the pseudo-code for contextualized learning in Appendix A. In practice, the distance measurement can be selected based on feature domain. For instance, in text domain, $dist(\cdot,\cdot)$ can be the cosine distance or the euclidean distance. For the refinement radius, we set r_j to be the p-th percentile value of $\{dist(x_{\lambda_j}, x_i)\}_{i=1}^n$, i.e., the set of all distances from each example $x_i \in U$ to the development data point x_{λ_j}, where p is a system hyperparameter that can be selected based on the validation accuracy of the resultant estimated soft labels. We note that the Nemo contextualized learning pipeline is compatible with any label modeling approach, where the LF contextualizer is essentially a pre-processing step on the weak supervision sources and the final probabilistic soft labels can be learned using any user-specified label model. This design allows Nemo to flexibly benefits from the advance in weak supervision modeling approaches, where the direction has received increasing research attention.

Example 4.6. In Figure 7, if we leverage the context that λ_1 and λ_3 were created from the red/blue-start points respectively, contextualized learning pipeline would be able to refine the LFs, and hopefully resolve the conflicts perfectly.

5 EVALUATION

We evaluate the end-to-end performance of Nemo, and perform a suite of ablation studies on (1) the proposed SEU selection strategy and (2) the contextualized learning approach. We seek to validate the following claims in response to the key hypotheses made:

- **Nemo makes the end-to-end DP learning paradigm more efficient and productive.** In Section 5.2, we see that Nemo much improves over the current DP workflow by an average 20%, validating that IDP formalism could enable optimization on DP learning paradigm.

- **The proposed selection strategy SEU improves the efficiency of LF development over existing selection methods.** In Section 5.3, we see that SEU offers significant performance lift over the random sampling baseline by an average 16%, validating that intelligent selection method can indeed improve the efficiency of DP pipeline and the effectiveness of SEU for this selection problem.
- **The proposed contextualized learning pipeline improves the learning performance over the standard learning pipeline.** In Section 5.4, we see that the contextualized learning pipeline leads to a considerable lift over standard learning pipeline by an average 11%, validating the importance of exploiting LF development context in learning and the effectiveness of the proposed approach.

5.1 Evaluation Setup

Datasets. We conduct experiments across five textual datasets and one image dataset, spanning three different tasks: sentiment classification, spam classification, and visual relation classification. For sentiment classification, we include Amazon Review [15], Yelp Review [43], and IMDB Review [22]. For spam classification, we include Youtube [1] and SMS [2]. For visual relation classification, we use the Visual Genome dataset [19]. For each dataset, we randomly partition the data into 80% training set, 10% validation set, and 10% test set, following the common convention [41]. We provide the dataset statistics in Table 1. Note that we only use a subset of the Visual Genome dataset that contains examples with the visual relationships of interest. Specifically, we formulate a binary classification task where the model is to classify whether an image contains the visual relationship of "carrying" or "riding".

Evaluation Protocol. For performance comparisons, similar to active learning setting, we plot out the *learning curve* of end model generalization performance on the test set over the number of iterations. A more efficient method could achieve higher performance within fewer number of iterations. We measure the generalization performance using *Accuracy Score* for all datasets except for SMS, in which we use *F1-score* since the dataset is highly imbalanced. For ease of comparisons, we summarize each learning curve by calculating the average performance on the learning curve, which essentially corresponds to its *area under curve*. Formally, given a learning curve represented by a set of points $C = \{(x_0, y_0), \ldots, (x_n, y_n)\}$ where $x_{i-1} < x_i$ and each (x_i, y_i) corresponds to the model performance y_i after x_i iterations, we summarize the performance of the curve by: $\frac{1}{n}\sum_{i=1}^{n} y_i$. In the experiments, we perform a total of 50 iterations and evaluate the model performance every 5 iterations. We include the evaluation plots in Appendix B. All reported results are the averaged over 5 runs with different random initializations.

Simulated User. In the experiments, apart from the user study conducted in Section 5.2, we leverage simulated user to allow more extensive evaluations of the methods. Similar to [6], we utilize ground truth labels to simulate real user feedbacks. Specifically, when an example x_i is selected and x_i contains a set of primitives $C = \{z \in \mathcal{Z} | z \text{ contained in } x_i\}$, we first build a set of candidate LFs $\Lambda = \{\lambda_{z,y_i} | z \in C\}$. Then, to resemble human expertise, the candidate set Λ is refined by filtering out LFs whose accuracy is lower than some threshold t. Finally, one LF is sampled from the refined set of candidate LFs and returned [1]. We set $t = 0.5$ in the experiments if not otherwise mentioned.

Table 1: Dataset statistics. "Cls." stands for classification task.

Task	Dataset	#Train	#Valid	#Test
Sentiment Cls.	Amazon	14,400	1,800	1,800
	Yelp	20,000	2,500	2,500
	IMDB	20,000	2,500	2,500
Spam Cls.	Youtube	1,566	195	195
	SMS	4,458	557	557
Visual Relation Cls.	VG	5,084	635	635

Implementation Details. In the experiments, we featurize the input text examples with their TF-IDF representation for text datasets, and use pre-trained Resnet [14] to extract feature for image examples. We fix the end model to be logistic regression model for all methods. If not otherwise mentioned, we adopt MeTaL [30] as the underlying label model to aggregate the weak labels. We consider the primitive domain $\mathcal{Z}$ to be the set of uni-grams in training examples for text datasets. For the image dataset, we utilize the associated object annotations as the corresponding primitives for each image. In practice, such object annotations can be obtained by leveraging any off-the-shelf object detection models. We include more details in Appendix C.

5.2 Nemo End-to-End System Performance

We demonstrate that Nemo outperforms existing baseline methods across various datasets through extensive quantitative experiments and an user study. We validate the importance of different Nemo system components, and demonstrate Nemo's robustness to the change of LFs' accuracy level.

Comparisons to baseline methods. We include two sets of baseline methods in our experiments. First, we include existing methods that are subsumed under IDP framework:

- *Snorkel* [28]: Snorkel is a vanilla instantiation of IDP that selects development data by random sampling, and learns from LFs without utilizing their development context. It is the predominant approach used in practice.
- *Snorkel-Abs* [9]: Snorkel-Abs is a selection-only IDP method that adaptively selects development data on which the current LFs abstain the most. It learns from LFs without considering their development context.
- *Snorkel-Dis* [9]: Snorkel-Dis is as well a selection-only IDP approach that stratigically selects development data on which the current LFs disagree the most. It does not leverage LF development context in learning.
- *ImplyLoss-L* [2] [3]: ImplyLoss-L is a contextualized-learning only IDP approach. It learns from LFs with their development context modeled through a deliberately designed model and loss function. However, it does not consider strategically selecting the development data. Thus, we couple it with the random sampling selection method.

[1] When available, we leverage external lexicon (e.g., opinion lexicon for sentiment [18]) to further simulate real user responses. More details in Appendix C.

[2] Note that we modify the discriminative part of the ImplyLoss model to be a linear model (hence the suffix "L") for consistency across the methods.

Table 2: Performances of Nemo and existing baselines across datasets. We see Nemo consistently outperforms the baselines, and that the proposed IDP framework offers strong performance when compared to other existing interactive schemes.

	Methods								
	Full IDP	Vanilla IDP	Selection-only IDP		CL-only IDP	Other Interactive Schemes			
Dataset	Nemo	Snorkel	Snorkel-Abs	Snorkel-Dis	ImplyLoss-L	US	IWS-LSE	BALD	AW
Amazon	**0.7674**	0.6774	0.6783	0.6733	0.6822	0.5970	0.6234	0.6193	0.6951
Yelp	**0.7907**	0.6556	0.6664	0.6887	0.7009	0.6239	0.6415	0.6129	0.6745
IMDB	**0.7958**	0.7107	0.7338	0.7480	0.6766	0.6058	0.6295	0.5933	0.7247
Youtube	**0.8722**	0.8235	0.8541	0.8527	0.6811	0.7609	0.7904	0.7816	0.8073
SMS	**0.7038**	0.4789	0.6189	0.5485	0.5065	0.4234	0.6305	0.4536	0.5569
VG	**0.6701**	0.6152	0.6250	0.6384	0.6270	0.5662	0.5976	0.5703	0.5914

Table 3: Performances and median user response time for different methods in the user study on Amazon dataset. Nemo significantly outperforms the baselines while taking slightly more time for users to respond compared to US and IWS-LSE.

	Methods						
	Full IDP	Vanilla IDP	Selection-only IDP		CL-only IDP	Other Interactive Schemes	
Metric	Nemo	Snorkel	Snorkel-Abs	Snorkel-Dis	ImplyLoss-L	US	IWS-LSE
Performance	**0.7473**	0.6665	0.6689	0.6600	0.6833	0.5882	0.5971
React Time (Median)	14.42s	16.21s	17.95s	13.05s	16.21s	12.50s	**6.73s**

Second, we include methods under other related interactive schemes:

- *Uncertainty Sampling (US)* [20]: US is a classic and competitive method within active learning paradigm [34].
- *BALD* [12, 17]: BALD is a representative bayesian active learning method that has been used as a strong active learning baseline in recent literature.
- *IWS-LSE* [6]: IWS-LSE is a representative method under the interactive weak supervision paradigm considered in [6, 13], where the user is iteratively queried to provide feedback on whether a suggested labeling heuristic is useful or not.
- *Active Weasul (AW)* [5]: AW combines active learning with weak supervision by asking the user to hand-label selected data points in order to help the label model in denoising a *fixed set* of LFs. Note that AW requires an initial given set of LFs. In the experiments, we use Snorkel in the first 10 iterations to generate the LF set.

In Table 2, we see that Nemo consistently outperforms all the baseline methods by a significant margin, with an average 9% performance lift over the second-best performing methods in each dataset. More closely, by comparing Nemo to Snorkel, we observe an average 20% improvement, verifying the importance and benefits of the IDP framework that considers optimizing and exploiting LF development process in the DP pipeline. While Snorkel-Abs and Snorkel-Dis improve over vanilla Snorkel by considering more adaptive data selection methods, the performance lift is much smaller in these cases. This is largely due to the heuristic design of the two approaches and their lack of further leveraging LF development context in learning. For ImplyLoss-L, although it improves over Snorkel (in 3 out of 5 datasets) by learning with contextualized LFs, without strategically selecting the development data, its performance lift appears limited especially when compared to Nemo. In sum, while previous methods had improved over the vanilla Snorkel baseline by either (1) designing better selection strategies or (2) learning with LF development context, none of these methods have considered both problems in an unifying interactive framework, thus rendering limited performance improvement when compared to Nemo which tackles both problems simultaneously in IDP.

From Table 2, we also observe that by soliciting user supervision at functional level, i.e., LFs, IDP methods (including the vanilla Snorkel) achieve better performances over US that queries for user feedback at single label annotation level, by up to 66% improvement. Similarly, IDP methods generally perform preferably against IWS-LSE that queries for user feedback on the usefulness of single labeling heuristic per iteration. Notably, the real user effort spent in answering each query in different interactive schemes may not be completely reflected by the reported performances, since varying interactive schemes require different types of user inputs. However, we believe that the results indeed demonstrate the importance and the promising potential of studying the proposed IDP framework.

Case study with real users. We conduct a carefully designed user study to validate the effectiveness of Nemo with real users developing the LFs based on selected development data. In the user study, we invited a total of 15 users with knowledge in basic machine learning as participants, including graduate students and industry engineers working in related fields. In the study, each user is asked to perform 2 randomly assigned interactive learning tasks, so that each method receives results from 5 users. We compute the result for ImplyLoss based on LFs created in the Snorkel user study, since random selection strategy is used in both approaches. In each task, the user goes through a total of 30 interactive iterations, and we record the model performance every 3 iterations. We randomly shuffle the order an user perform on the 2 assigned tasks to avoid potential bias that could be introduced by the ordering of different methods. Before the start of each study, we gave short introduction about the background of Data Programming.

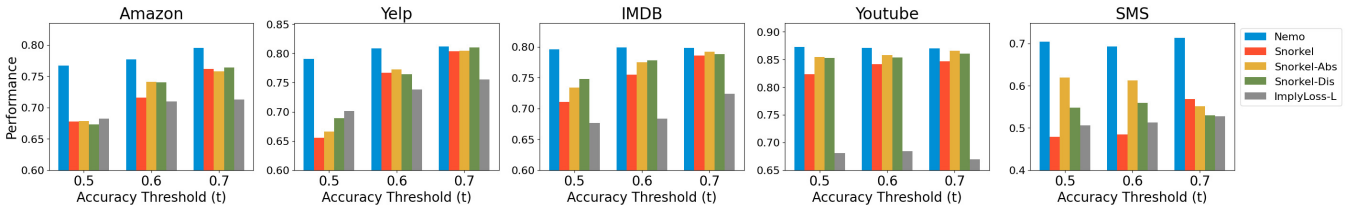

Figure 8: Performances under varying LF accuracy thresholds. Nemo demonstrates the strongest robustness when LF accuracy threshold decreases from 0.7 to 0.5.

Table 4: Comparisons between Nemo performance with and without either the data selector or the LF contextualizer. We see that both components are critical to Nemo.

		Ablated Version	
Dataset	Nemo	No Data Selector	No LF Contextualizer
Amazon	**0.7674**	0.7244	0.7384
Yelp	**0.7907**	0.7360	0.7219
IMDB	**0.7958**	0.7557	0.7932
Youtube	**0.8722**	0.8407	0.8628
SMS	**0.7038**	0.6092	0.6899
VG	**0.6701**	0.6253	0.6542

We perform the study using the sentiment classification task on Amazon Review dataset which most users are familiar with. We report the user study results in Table 3. We observe an overall similar trend to our findings with simulated users. Specifically, Nemo provides a significant performance lift, up to 13% and 27%, over existing DP and interactive methods respectively. This showcases that Nemo, by supporting the full IDP loop, indeed leads to an efficient and productive learning workflow in practice with real users. In addition, we see that users generally spend a little more time in providing LFs as feedback in IDP setting than providing label annotation as response in active learning setting. The overhead lies in the extra time, which is roughly around 2 to 3 seconds per iteration, that an user needs to select a corresponding primitive from the development data in addition to determining its ground truth label. We also note that IWS-LSE has the shortest user response time, as determining whether an LF is useful or not is generally easier than determining the label for a data point.

Ablation study on Nemo. We study the importance of the two core components in Nemo, i.e., the development data selector and the LF contextualizer. Specifically, we evaluate Nemo's performance when either one of the components is removed. We see that in Table 4, removing either components decreases Nemo's performance, with an average 7% and 3% drop when the data selector and the LF contextualizer is removed respectively. This showcases the importance of both components in Nemo and the need to consider both the data selection and contextualized learning problems in IDP.

Sensitivity to LF precision. We evaluate the robustness of Nemo with respect to the accuracy of input LFs. Recall that when simulating real user response, the oracle simulator filters out LFs with accuracy lower than a given threshold t. Here, we evaluate Nemo under varying values of t, along with comparisons to other baseline IDP methods. From Figure 8, we first observe that as the threshold value increases, there is an overall trend of performance improvements for all methods, suggesting that, perhaps unsurprisingly, users could in general improve DP learning performance by providing more accurate LFs. Next, we see that regardless of the accuracy threshold values, Nemo consistently achieves the best performance as compared to other baselines. Finally, Nemo demonstrate stronger robustness to the threshold value change than the baseline methods. For example, we see that the performance of Snorkel drastically drops when the threshold value decreases from 0.7 to 0.5, whereas Nemo remains much stable across different threshold values. The results suggest that Nemo could be more reliably deployed in practice where the accuracies of LFs developed could vary in range.

Table 5: Performances of different selection strategies when the learning pipeline is fixed to be the standard vanilla approach without using LF development context. We see that SEU consistently outperforms, by a large margin, the other baselines in all datasets considered.

	Selection Strategy			
Dataset	SEU	Random	Abstain	Disagree
Amazon	**0.7384**	0.6774	0.6783	0.6733
Yelp	**0.7219**	0.6556	0.6664	0.6887
IMDB	**0.7932**	0.7107	0.7338	0.7480
Youtube	**0.8628**	0.8235	0.8541	0.8527
SMS	**0.6899**	0.4789	0.6189	0.5485
VG	**0.6542**	0.6152	0.6250	0.6384

5.3 SEU Selection Strategy Performance

We evaluate the effectiveness of the proposed selection strategy SEU by comparing it to different baseline selection approaches, and ablating various aspects of its design. To focus on the comparisons between different selection strategies alone, we fix the learning pipeline to be the standard vanilla procedure (without the use of LF development context) in the following sets of experiments.

Comparisons to baseline selection approaches. We compare SEU to three baseline selection methods:

- The *Random* baseline that selects randomly from the unlabeled set [28].
- The *Abstain* baseline that selects the data point on which the current LFs abstain the most [9].
- The *Disagree* baseline that selects the data point on which the current LFs disagree the most [9].

From Table 5, we see that SEU consistently enjoys better performance than the other baseline selection approaches often by a large

Table 6: Ablation study on the SEU's user model. We see that the accuracy-weighted design is critical to SEU's success.

	User Model	
Dataset	SEU (Eq. 6)	Uniform
Amazon	**0.7384**	0.6774
Yelp	**0.7219**	0.6556
IMDB	**0.7932**	0.7107
Youtube	**0.8628**	0.8235
SMS	**0.6899**	0.4789
VG	**0.6542**	0.5592

Table 7: Ablation study on SEU's LF utility function. We see that SEU achieves best performances when the utility function captures both the informativeness and the correctness aspects of an LF.

	LF Utility Function		
Dataset	SEU (Eq. 3)	No Informativeness	No Correctness
Amazon	**0.7384**	0.7369	0.6683
Yelp	**0.7219**	0.7211	0.6536
IMDB	**0.7932**	0.7911	0.7847
Youtube	**0.8628**	0.8538	0.8552
SMS	**0.6899**	0.6695	0.6517
VG	**0.6542**	0.6486	0.6346

margin. When compared to Random, SEU provides performance lift up to 44%, validating its capability of strategically selecting useful development data to guide efficient development of LFs. When compared to Abstain and Disagree, SEU as well enjoys improvement up to 26%, demonstrating its advantage over these existing straightforward strategies.

Ablation study on user model. Recall that in SEU's user model (Eq. 6), the conditional probability $P(\lambda_{z,y}|x)$, if not zero, is modeled based on the estimated accuracy of $\lambda_{z,y}$. SEU models that an user would have higher probability of picking a primitive z from x to create an LF $\lambda_{z,y}$ if $\lambda_{z,y}$ has higher accuracy. Here, we examine the importance of this accuracy-weighted design in the user model by comparing to a baseline where we modify the user model to *not consider* the accuracies of LFs and to essentially assign *uniform* probability to the potential LFs $\{\lambda_{z,y}|z \in \mathcal{Z}, z \text{ contained in } x\}$:

$$P(\lambda_{z,y}|x) = \begin{cases} P(y) \cdot \frac{1}{\sum_{\lambda \in \{\lambda_{z,y}|z \text{ in } x\}} 1}, & \text{if } z \text{ contained in } x \\ 0, & \text{otherwise} \end{cases}$$

From Table 6, we see that it is indeed important to model $P(\lambda_{z,y}|x)$ differently based on the accuracy of $\lambda_{z,y}$, where the performance of SEU much degrades when $P(\lambda_{z,y}|x)$ is modeled uniformly for all possible LFs.

Ablation study on LF utility function. Another core component in SEU is the LF utility function, which is designed to assign higher score to LFs that provide informative and accurate supervision signals. Specifically, recall that in Eq. 3, the first term $\psi_t^{\text{uncertainty}}(x_i)$ and the second term $\lambda(x_i)\hat{y}_i$ capture the informativeness and the correctness of the provided label $\lambda(x_i)$ respectively. We evaluate the importance of both aspects by comparing to two ablated versions of utility functions, in which we remove either term respectively:

Table 8: Performances of different approaches to learn from LFs. We see that contextualized pipeline improves over the standard pipeline and the ImplyLoss model.

	Learning Approach		
Dataset	Contextualized	Standard	ImplyLoss
Amazon	**0.7244**	0.6774	0.6822
Yelp	**0.7360**	0.6556	0.7009
IMDB	**0.7557**	0.7107	0.6766
Youtube	**0.8407**	0.8235	0.6811
SMS	**0.6092**	0.4789	0.5065
VG	0.6253	0.6152	**0.6270**

- No Informativeness: $\Psi_t(\lambda) = \sum_{i \in C} \lambda(x_i)\hat{y}_i$
- No Correctness: $\Psi_t(\lambda) = \sum_{i \in C} \psi_t^{\text{uncertainty}}(x_i)$

In Table 7, we see that removing either the informativeness or the correctness term in the utility function decreases SEU performance, suggesting the importance of considering both properties when measuring an LF's utility.

5.4 Contextualized Learning Performance

We evaluate the effectiveness of the proposed contextualized learning pipeline, and ablate different aspects of its design choices. To focus on the comparisons between different approaches to learn from the LFs, we fix the data selection strategy to be the vanilla random selection approach in the following sets of experiments.

Comparisons to baseline methods. We compare the contextualized learning pipeline to two baseline approaches:

- The *Standard* learning pipeline that learns from the LFs without utilizing any LF contextual information.
- The *Implyloss* learning model that learns from the LFs and their contextual information through a specialized model and loss function.

In Table 8, we demonstrate that the proposed contextualized learning pipeline can effectively leverage the LF development context to better model and learn from the weak supervision sources. It improves over the standard learning pipeline by up to 27%. In particular, by only refining the LFs' coverage and learn with the same underlying label model (MeTaL), we obtain larger performance improvement compared to the improvement brought by designing a more sophisticated ImplyLoss model.

Ablation study on distance function. Recall that in the contextualized learning pipeline, the LF contextualizer relies on a distance function to refine the LFs. We compare how different distance measurements affect the performance. In Table 9, we see that Cosine distance generally brings larger performance lift than Euclidean distance. We note that regardless of the distance function used, the contextualized pipeline improves over the standard counterpart.

6 RELATED WORK

Recent progress in DP has largely been made in developing advanced label models for various applications [4, 8, 10, 11, 16, 25, 29, 30, 32, 33, 35, 39]. We defer readers to [40] for a more comprehensive survey on weak supervision methods.

Labeling Function Development. To reduce user effort spent in LF development, studies have mainly taken three directions:

Table 9: Contextualized learning with different distance functions. Cosine distance offers larger lift than Euclidean distance while both improves over standard pipeline.

	Contextualized		
Dataset	Cosine Distance	Euclidean Distance	Standard
Amazon	**0.7244**	0.6913	0.6774
Yelp	**0.7360**	0.6991	0.6556
IMDB	**0.7557**	0.7200	0.7107
Youtube	**0.8407**	0.8181	0.8235
SMS	0.6092	**0.6174**	0.4789
VG	0.6253	**0.6332**	0.6152

(1) *automatic LF generation*, (2) *interactive LF discovery*, and (3) *interactively-guided LF development* which our work falls into. Automatic LF generation methods aim to create LFs automatically. The methods generally require an initial set of labeled data, or seed LFs developed by users. Snuba [38] learns weak classifiers as heuristic models from a small labeled dataset; TALLOR [21] and GLaRA [44] use an initial set of seed LFs to generate new ones by compounding multiple simpler LFs and by exploiting the semantic relationship of the seed LFs respectively; [36] applies program systhesis to generate task-level LFs from a set of labeled data and domain-level LFs. Interactive LF discovery methods consider adaptively searching for useful LFs from a large candidate set, by interactively querying for user's feedback on whether some suggested LFs are useful or not. The candidate LFs are generated based on context-free grammar [13], n-grams [6], or pretrained language models [42]. Based on the user's feedback on the usefulness (whether an LF is better than random) of some selected LFs, the systems adapt and learn to identify promising LFs from the large candidate set, which are output as the final LFs to be used in the subsequent DP pipeline. Unlike the above two directions that require different forms of user inputs in LF development process as compared to the existing workflow, our work takes the third direction (interactively-guided LF development), which is built to inherently support the existing workflow used in practice, i.e., users write LFs by drawing ideas from development data. However, instead of simply selecting the development data randomly from the unlabeled set, this direction considers strategically selecting informative development data to guide the users in designing useful LFs efficiently. Within this direction, [9] performed an initial exploration, but in a relatively ad-hoc way. Our work proposes the first formalism, IDP, for the problem that further extends the scope to exploiting the information in LF development process to better model the resultant LFs.

Connecting Data Programming and Active Learning. Related to our work, prior studies have explored the connection between DP and active learning from other perspectives: (1) applying active learning to complement DP, and (2) leveraging DP techniques for active learning. Within the first direction, [23] proposed to complement an existing set of LFs by asking users to annotate a selected subset of unlabeled data; Similarly, [5] asks users to label selected data points that would be most informative in helping denoise and aggregate the weak supervision sources in DP; Asterisk [26] employed an active learning process to enhance the quality, in terms of accuracy and coverage, of the weak labels initially provided by the LFs. On the second direction, [27] applied DP to generate an initial set of weak labels to improve the efficiency of a later active learning process; [24] aimed to expand an initial set of labeled data with examples from another larger unlabeled dataset. The method first constructs neighborhood-based LFs from the seed data, and utilizes the LFs to identify relevant candidate examples from the larger unlabeled set, where the candidate examples are finally annotated by application users.

7 DISCUSSION

While we have primarily focused on the atomic one-example to one-LF IDP setup, in this section, we discuss how Nemo can be flexibly extended to support the general setup wherein users can leverage multiple examples to create multiple LFs in each IDP iteration. Specifically, the extension includes (1) a system-level feature that allows users to use multiple examples when creating an LF, and (2) an algorithmic-level redesign that enables Nemo to model the probability of the user returning multiple LFs per iteration.

Primitive-based Example Explorer. One limitation on the atomic IDP setup is that users might sometimes find it hard to develop a new LF by looking at a single example, as it may be difficult to judge how well an LF can generalize to other examples. As a result, we enrich the Nemo user interface with the *primitive-based example explorer*. As shown in Figure 5, when shown a development data point, the user can click on the candidate primitives and utilize the example explorer to view a randomly sampled set of examples containing the selected primitive. In this way, the user is able to leverage additional data points to evaluate the quality of an LF.

Multi-LF User Model. We demonstrate how SEU can be generalized to accommodate the IDP setup where the user may return multiple LFs in each iteration. First, we modify the SEU utility measurement (Eq. 1) to the following:

$$x^* = \arg\max_{x \in U} \mathbb{E}_{P(\Lambda|x)}[\sum_{\lambda \in \Lambda} \Psi_t(\lambda)], \tag{5}$$

Note that we replace λ (a single LF) with Λ (a set of LFs). In addition, we measure the usefulness of the returned set of LFs by summing over the individual utilities of the included LFs. Then, we can measure the user model by $P(\Lambda|x) = \Pi_{\lambda \in \Lambda} P(\lambda|x)$, where:

$$P(\lambda_{z,y}|x) = \begin{cases} P(y) \cdot acc(\lambda_{z,y}) \cdot \mathbb{1}_{acc(\lambda_{z,y})>0.5}, & \text{if } z \text{ contained in } x \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Indeed, these are only two examples on how one can augment Nemo to support different IDP workflows in practice. We look forward to seeing more future work that explores other possibilities.

8 CONCLUSION

We formalize IDP where LF development is considered a central component in the entire DP learning pipeline. In IDP, we study how to strategically select development data for guiding LF development, and how to exploit the development context for better LF modeling. We then introduce Nemo for IDP, which is built upon the novel SEU selection strategy and the contextualized learning pipeline. We validate that Nemo leads to more efficient and productive DP pipeline over the existing prevailing workflow, with an average 20% (and up to 47%) improvement across various datasets and tasks.

REFERENCES

[1] T. C. Alberto, J. V. Lochter, and T. A. Almeida. 2015. TubeSpam: Comment Spam Filtering on YouTube. In *ICMLA*. 138–143. https://doi.org/10.1109/ICMLA.2015.37

[2] Tiago A Almeida, José María G Hidalgo, and Akebo Yamakami. 2011. Contributions to the study of SMS spam filtering: new collection and results. In *DocEng*. 259–262.

[3] Abhijeet Awasthi, Sabyasachi Ghosh, Rasna Goyal, and Sunita Sarawagi. 2020. Learning from Rules Generalizing Labeled Exemplars. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SkeuexBtDr

[4] Stephen H Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, et al. 2019. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *SIGMOD (Industrial)*. 362–375.

[5] Samantha Biegel, Rafah El-Khatib, Luiz Otavio Vilas Boas Oliveira, Max Baak, and Nanne Aben. 2021. Active WeaSuL: Improving Weak Supervision with Active Learning. *arXiv preprint arXiv:2104.14847* (2021).

[6] Benedikt Boecking, Willie Neiswanger, Eric Xing, and Artur Dubrawski. 2021. Interactive Weak Supervision: Learning Useful Heuristics for Data Labeling. In *International Conference on Learning Representations*. https://openreview.net/forum?id=IDFQI9OY6K

[7] Eran Bringer, Abraham Israeli, Yoav Shoham, Alexander J. Ratner, and Christopher Ré. 2019. Osprey: Weak Supervision of Imbalanced Extraction Problems without Code. *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning* (2019).

[8] Salva Rühling Cachay, Benedikt Boecking, and Artur Dubrawski. 2021. End-to-End Weak Supervision. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=gbcsmD3Iznu

[9] Benjamin Cohen-Wang, Steve Mussmann, Alexander Ratner, and Christopher Ré. 2019. Interactive Programmatic Labeling for Weak Supervision. *KDD Data Collection, Curation, and Labeling for Mining and Learning Workshop* (2019).

[10] Nilaksh Das, Sanya Chaba, Renzhi Wu, Sakshi Gandhi, Duen Horng Chau, and Xu Chu. 2020. Goggles: Automatic image labeling with affinity coding. In *SIGMOD*. 1717–1732.

[11] Daniel Fu, Mayee Chen, Frederic Sala, Sarah Hooper, Kayvon Fatahalian, and Christopher Ré. 2020. Fast and three-rious: Speeding up weak supervision with triplet methods. In *International Conference on Machine Learning*. PMLR, 3280–3291.

[12] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. 2017. Deep bayesian active learning with image data. In *International Conference on Machine Learning*. PMLR, 1183–1192.

[13] Sainyam Galhotra, Behzad Golshan, and Wang-Chiew Tan. 2021. Adaptive rule discovery for labeling text data. In *Proceedings of the 2021 International Conference on Management of Data*. 2217–2225.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[15] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[16] Sarah Hooper, Michael Wornow, Ying Hang Seah, Peter Kellman, Hui Xue, Frederic Sala, Curtis Langlotz, and Christopher Re. 2020. Cut out the annotator, keep the cutout: better segmentation with weak supervision. In *ICLR*.

[17] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. 2011. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745* (2011).

[18] Minqing Hu and B. Liu. 2004. Mining and summarizing customer reviews. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004).

[19] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. 2017. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision* 123, 1 (2017), 32–73.

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

[21] Jiacheng Li, Haibo Ding, Jingbo Shang, Julian McAuley, and Zhe Feng. 2021. Weakly Supervised Named Entity Tagging with Learnable Logical Rules. In *ACL*. 4568–4581. https://doi.org/10.18653/v1/2021.acl-long.352

[22] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 142–150.

[23] Ayush Maheshwari, Oishik Chatterjee, Krishnateja Killamsetty, Ganesh Ramakrishnan, and Rishabh K. Iyer. 2021. Semi-Supervised Data Programming with Subset Selection. In *FINDINGS*.

[24] Neil Rohit Mallinar, Abhishek Shah, T. Ho, Rajendra Ugrani, and Ayushi Gupta. 2020. Iterative Data Programming for Expanding Text Classification Corpora. *ArXiv* abs/2002.01412 (2020).

[25] Alessio Mazzetto, Cyrus Cousins, Dylan Sam, Stephen H Bach, and Eli Upfal. 2021. Adversarial Multi Class Learning under Weak Supervision with Performance Guarantees. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, 7534–7543. https://proceedings.mlr.press/v139/mazzetto21a.html

[26] Mona Nashaat, Aindrila Ghosh, James Miller, and Shaikh Quader. 2020. Asterisk: Generating Large Training Datasets with Automatic Active Supervision. *ACM Transactions on Data Science* 1, 2 (2020), 1–25.

[27] Mona Nashaat, Aindrila Ghosh, James Miller, Shaikh Quader, Chad Marston, and J. Puget. 2018. Hybridization of Active Learning and Data Programming for Labeling Large Industrial Datasets. *2018 IEEE International Conference on Big Data (Big Data)* (2018), 46–55.

[28] 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.

[29] Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems* 29 (2016), 3567.

[30] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. 2019. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4763–4771.

[31] Christopher Ré, Feng Niu, Pallavi Gudipati, and Charles Srisuwananukorn. 2019. Overton: A data system for monitoring and improving machine-learned products. *arXiv preprint arXiv:1909.05372* (2019).

[32] Wendi Ren, Yinghao Li, Hanting Su, David Kartchner, Cassie Mitchell, and Chao Zhang. 2020. Denoising Multi-Source Weak Supervision for Neural Text Classification. In *Findings of EMNLP*.

[33] Esteban Safranchik, Shiying Luo, and Stephen Bach. 2020. Weakly supervised sequence tagging from noisy rules. In *AAAI*, Vol. 34. 5570–5578.

[34] Burr Settles. 2009. Active learning literature survey. (2009).

[35] Changho Shin, Winfred Li, Harit Vishwakarma, Nicholas Roberts, and Frederic Sala. 2022. Universalizing Weak Supervision. In *ICLR*.

[36] Albert Tseng, Jennifer J Sun, and Yisong Yue. 2021. Automatic Synthesis of Diverse Weak Supervision Sources for Behavior Analysis. *arXiv preprint arXiv:2111.15186* (2021).

[37] Paroma Varma, Bryan He, Payal Bajaj, Imon Banerjee, Nishith Khandwala, Daniel L Rubin, and Christopher Ré. 2017. Inferring generative model structure with static analysis. *Advances in neural information processing systems* 30 (2017), 239.

[38] Paroma Varma and Christopher Ré. 2018. Snuba: Automating weak supervision to label training data. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 12. NIH Public Access, 223.

[39] Paroma Varma, Frederic Sala, Shiori Sagawa, Jason Fries, Daniel Fu, Saelig Khattar, Ashwini Ramamoorthy, Ke Xiao, Kayvon Fatahalian, James Priest, and Christopher Ré. 2019. Multi-Resolution Weak Supervision for Sequential Data. In *NeurIPS*, Vol. 32. https://proceedings.neurips.cc/paper/2019/file/93db85ed909c13838ff95ccfa94cebd9-Paper.pdf

[40] Jieyu Zhang, Cheng-Yu Hsieh, Yue Yu, Chao Zhang, and Alexander Ratner. 2022. A Survey on Programmatic Weak Supervision. *arXiv preprint arXiv:2202.05433* (2022).

[41] Jieyu Zhang, Yue Yu, Yinghao Li, Yujing Wang, Yaming Yang, Mao Yang, and Alexander Ratner. 2021. WRENCH: A Comprehensive Benchmark for Weak Supervision. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. https://openreview.net/forum?id=Q9SKS5k8io

[42] Rongzhi Zhang, Yue Yu, Pranav Shetty, Le Song, and Chao Zhang. 2022. PRBoost: Prompt-Based Rule Discovery and Boosting for Interactive Weakly-Supervised Learning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[43] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems* 28 (2015), 649–657.

[44] Xinyan Zhao, Haibo Ding, and Zhe Feng. 2021. GLaRA: Graph-based Labeling Rule Augmentation for Weakly Supervised Named Entity Recognition. In *EACL*. 3636–3649. https://aclanthology.org/2021.eacl-main.318