HHS Public Access

Author manuscript

Adv Neural Inf Process Syst. Author manuscript; available in PMC 2019 December 23.

Published in final edited form as:

Adv Neural Inf Process Syst. 2019 December ; 32: 9392–9402.

Slice-based Learning: A Programming Model for Residual Learning in Critical Data Slices

Vincent S. Chen, **Sen Wu**, **Zhenzhen Weng**, **Alexander Ratner**, **Christopher Ré**
Stanford University

Abstract

In real-world machine learning applications, data subsets correspond to especially critical outcomes: vulnerable cyclist detections are safety-critical in an autonomous driving task, and "question" sentences might be important to a dialogue agent's language understanding for product purposes. While machine learning models can achieve high quality performance on coarse-grained metrics like F1-score and overall accuracy, they may underperform on critical subsets—we define these as *slices*, the key abstraction in our approach. To address slice-level performance, practitioners often train separate "expert" models on slice subsets or use multi-task hard parameter sharing. We propose *Slice-based Learning*, a new programming model in which the *slicing function (SF)*, a programming interface, specifies critical data subsets for which the model should commit additional capacity. Any model can leverage SFs to learn *slice expert representations*, which are combined with an attention mechanism to make *slice-aware* predictions. We show that our approach maintains a parameter-efficient representation while improving over baselines by up to 19.0 F1 on slices and 4.6 F1 overall on datasets spanning language understanding (e.g. SuperGLUE), computer vision, and production-scale industrial systems.

1 Introduction

In real-world applications, some model outcomes are more important than others: for example, a data subset might correspond to safety-critical but rare scenarios in an autonomous driving setting (e.g. detecting cyclists or trolley cars [18]) or critical but lower-frequency healthcare demographics (e.g. handling younger patients with certain cancers). Traditional machine learning systems optimize for overall quality, which may be too coarse-grained; models that achieve high overall performance might produce unacceptable failure rates on *slices* of the data. In many production settings, the key challenge is to maintain overall model quality while improving slice-specific metrics.

To formalize this challenge, we introduce the notion of *slices*: application-critical data subsets, specified programmatically by machine learning practitioners, for which the we would like to improve model performance. This leads to three technical challenges:

- **Coping with Noise:** Defining slices precisely can be challenging. While engineers often have a clear intuition of a slice, typically as a result of an error

vincentsc@cs.stanford.edu.

analysis, translating that intuition into a machine-understandable description can be a challenging problem, e.g., *"the slice of data that contains a yellow light at dusk."* As a result, any method must be able to cope with imperfect, overlapping definitions of data slices, as specified by noisy or *weak supervision*.

- **Stable Improvement of the Model:** Given a description of a set of slices, we want to improve the prediction quality on each of the slices without hurting overall model performance. Often, these goals are in tension: in many baseline approaches, steps to improve the slice-specific model performance would degrade the overall model performance, and vice-versa.
- **Scalability:** There may be many slices. Indeed, in industrial deployments of slicing-based approaches, hundreds of slices are commonly introduced by engineers [29]. This suggests that any approach to slicing must be judicious with adding parameters as the number of slices grow.

To improve *fine-grained*, i.e. slice-specific, performance, an intuitive solution is to create a separate model for each slice. To produce a single prediction at test time, one often trains a *mixture of experts* model (MoE) [17]. However, with the growing size of ML models, MoE is often untenable due to runtime performance, as it could require training and deploying hundreds of large models—one for each slice. Another strategy draws from multi-task learning (MTL), in which slice-specific *task heads* are learned with hard-parameter sharing [7]. This approach is computationally efficient but may not effectively share training data across slices, leading to suboptimal performance. Moreover, in MTL, tasks are distinct, while in *Slice-based Learning*, a single *base task* is refined by related slice tasks.

We propose a novel programming model, called *Slice-based Learning*, in which practitioners provide slicing functions (SFs), a programming abstraction for heuristically targeting data subsets of interest. SFs coarsely map input data to slice indicators, which specify data subsets for which we should allocate additional model capacity. To improve slice-level performance, we introduce *slice-residual-attention modules* (SRAMs) that explicitly model *residuals* between slice-level and the overall task predictions. SRAMs are agnostic to the architecture of any neural network model that they are added to— which we refer to as the *backbone* model—and we demonstrate our approach on state-of-the-art text and image models. Using shared backbone parameters, our model initializes slice "expert" representations, which are associated with learning slice-membership indicators and class predictors for examples in a particular slice. Then, slice indicators and prediction confidences are used in an *attention-mechanism* to reweight and combine each slice expert representation based on learned residuals from the base representation. This produces a *slice-aware* featurizatino of the data, which can be used to make a final prediction.

Our work fits into an emerging class of programming models that sit on top of deep learning systems [18, 27]. We are the first to introduce and formalize *Slice-based Learning*, a key programming abstraction for improving ML models in real-world applications subject to slice-specific performance objectives. Using an independent error analysis for the recent GLUE natural language understanding benchmark tasks [35], by simply encoding the identified error categories as slices in our framework, we show that we can improve the

quality of state-of-the-art models by up to 4.6 F1 points, and we observe slice-specific improvements of up to 19.0 points. We also evaluate our system on autonomous vehicle data and show improvements up to 15.6 F1 points on context-dependent slices (i.e. presence of bus, traffic light, etc.) and 2.3 F1 points overall. Anecdotally, when deployed in production systems [29], *Slice-based Learning* provides a practical programming model with improvements of up to 40 F1 points in critical test-time slices. On the SuperGlue [34] benchmark, this procedure accounts for a 2.7 improvement in overall score using the same architecture as a state-of-the-art modeling result. In addition to the proposal of SRAMs, we perform an in-depth analysis to explain the mechanisms by which SRAMs improve quality. We validate the efficacy of quality and noise estimation in SRAMs and compare to weak supervision frameworks [27] that estimate the quality of supervision sources to improve overall model accuracy. We show that by using SRAMs, we are able to produce accurate quality estimates, which leads to higher downstream performance on such tasks by an average of 1.1 overall F1 points.

2 Related Work

Our work draws inspiration from three main areas: mixture of experts, multi-task learning, and weak supervision. Jacobs et. al [17] proposed a technique called **mixture of experts** that divides the data space into different homogeneous regions, *learns the regions of data separately*, and then combines results with a single gating network [33]. This work is a generalization of popular ensemble methods, which have been shown to improve predictive power by reducing overfitting, avoiding local optima, and combining representations to achieve optimal hypotheses [32]. We were motivated in part by reducing the runtime cost and parameter count for such models.

MTL models provide the flexibility of *modular* learning—specific task heads, layers, and representations can be changed in an application-specific, ad hoc manner. Furthermore, MTL models benefit from the computational efficiency and regularization afforded by hard parameter sharing [7]. There are often also performance gains seen from adding auxiliary tasks to improve representation learning objectives [8, 30]. While our approach draws high-level inspiration from MTL, we highlight key differences: whereas tasks are disjoint in MTL, slice tasks are formulated *micro-tasks* that are direct extensions of a base task—they are designed specifically to learn deviations from the base-task representation. In particular, sharing information, as seen in MTL cross-stitch networks [25], requires $\Omega(n^2)$ weights across n local tasks; our formulation only requires attention over $O(n)$ weights, as slice tasks operate on the *same* base task. For example, practitioners might specify yellow lights and night-time images as important slices; the model learns a series of micro-tasks—based solely on the data specification—to inform how its approach for the base task, object detection, should change in these settings. As a result, slice tasks are not fixed ahead of time by an MTL specification; instead, these micro-task boundaries are learned dynamically from corresponding data subsets. This style of information sharing is adjacent to cross-task knowledge in recent **multi-task learning** (MTL) models [31, 38], and we were inspired by these methods.

Weak supervision has been viewed as a new way to incorporate data of varying accuracy including domain experts, crowd sourcing, data augmentations, and external knowledge bases [2, 5, 6, 11, 13, 20, 24, 26, 28]. We take inspiration from *labeling functions* [28] in weak supervision as a programming paradigm, which has seen success in industrial deployments [2]. In weak supervision, a key challenge is to assess the accuracy of a training data point, which is a function of the sources that supervise it. In contrast, this work models this accuracy in a fine-grained manner, based on a learned representation—this leads to higher overall quality.

Weak supervision and multitask learning can be viewed as orthogonal to slicing: we have observed them used alongside *Slice-based Learning* in academic projects and industrial deployments [29].

3 Slice-based Learning

We propose *Slice-based Learning* as a programming model for training machine learning models where users specify important data subsets to improve model performance. We describe the core technical challenges that lead to our notion of *slice-residual-attention modules* (SRAMs).

3.1 Problem statement

To formalize the key challenges of slice-based learning, we introduce some basic terminology. In our *base task*, we use a supervised input, ($x \in X$, $y \in Y$), where the goal is to learn according to a standard loss function. In addition, the user provides a set of k functions called *slicing functions (SFs)*, $\{\lambda_1,\ldots,\lambda_k\}$, in which $\lambda_i: X \rightarrow \{0,1\}$. These SFs are not assumed to be perfectly accurate; for example, SFs may be based on noisy or *weak* supervision sources in functional form [28]. SFs can come from domain-specific heuristics, distant supervision sources, or other off-the-shelf models, as seen in Figure 2. Ultimately, the model's goal is to improve (or avoid damaging) the overall accuracy on the base task while improving the model on the specified slices.

Formally, each of k slices, denoted $s_{i=1,\ldots,k}$, is an unobserved, indicator random variable, and each user-specified SF, $\lambda_{i=1,\ldots,k}$ is a corresponding, noisy specification. Given an input tuple $(X, Y, \{\lambda_i\}_{i=1,\ldots,k})$ consisting of a dataset (X, Y), and k different user-defined SFs λ_i, our goal is to learn a model $f_{\hat{w}}(\cdot)$—i.e. estimate model parameters $\hat{w}$ —that predicts $P(Y \mid \{s_i\}_{i=1,\ldots,k}, X)$ with high average slice-specific accuracy without substantially degrading overall accuracy.

Example 1 A developer notices that their self-driving car is not detecting cyclists at night. Upon error analysis, they diagnose that their state-of-the-art object detection model, trained on an automobile detection dataset (X,Y) of images, is indeed underperforming on night and cyclist slices. They write two SFs: λ_1 to classify night vs. day, based on pixel intensity; and λ_2 to detect bicycles, which calls a pretrained object detector for a bicycle (with or without a rider). Given these SFs, the developer leverages Slice-based Learning to improve model performance on safety-critical subsets.

Our problem setup makes a key assumption: SFs may be *non-servable* during test-time—i.e, during inference, an SF may be unavailable because it is too expensive to compute or relies on private metadata [1]. In Example 1, the potentially expensive cyclist detection algorithm is non-servable at runtime. When our model is served at inference, *SFs are not necessary*, and we can rely on the model's *learned* indicators.

3.2 Model Architecture

The *Slice-based Learning* architecture has six components. The key intuition is that we will train a standard prediction model, which we call the *base task*. We then learn a representation for each slice that explains how its predictions should differ from the representation of the base task—i.e., a *residual*. An attention mechanism then combines these representations to make a slice-aware prediction.

With this intuition in mind, the six components (Figure 2) are: (a) a **backbone**, (b) a set of k **slice-indicator heads**, and (c) k corresponding **slice expert representations**, (d) a **shared slice prediction head**, (e) a combined, **slice-aware representation**, and (f) a **prediction head**. Each SRAM operates over any backbone architecture and represents a path through components (b) through (e). We describe the architecture assuming a binary classification task (output dim. $c = 1$):

(a) **Backbone:** Our approach is agnostic to the neural network architecture, which we call the *backbone*, denoted $f_{\hat{w}}$, which is used primarily for feature extraction (e.g. the latest transformer for textual data, CNN for image data). The backbone maps data points x to a representation $z \in \mathbb{R}^d$.

(b) **Slice indicator heads:** For each slice, an indicator head will output an input's slice membership. The model will later use this to reweight the "expert" slice representations based on the likelihood that an example is in the corresponding slice. Each indicator head maps the backbone representation, z, to a logit indicating slice-membership: $\{q_i\}_{i=1,\ldots,k} \in \{0,1\}$ Each slice indicator head is supervised by the output of a corresponding SF, λ_i. For each example, we minimize the multi-label binary cross entropy loss $(\mathscr{L}_{\text{CE}})$ between the unnormalized logit output of each $q_i \lambda_i : \ell_{\text{ind}} = \sum_i^k \mathscr{L}_{\text{CE}}(q_i, \lambda_i)$

(c) **Slice expert representations:** Each slice representation, $\{r_i\}_{i=1,\ldots,k}$, will be treated as an "expert" feature for a given slice. We learn a linear mapping from the backbone, z, to each $r_i \in \mathbb{R}^h$, where h is the size of all slice expert representations.

(d) **Shared slice prediction head:** A shared, slice prediction head, $g(\cdot)$, maps each slice expert representation, r_i, to a logit, $\{p_i\}_{i=1,\ldots,k}$, in the output space of the base task: $g(r_i) = p_i \in \mathbb{R}^c$. where $c = 1$ for binary classification. We train slice "expert" tasks using *only* examples belonging to the corresponding slice, as specified by λ_i. Because parameters in $g(\cdot)$ are shared, each representation, r_i, is forced to specialize to the data belonging to its slice. We use the base task's

ground truth label, y, to train this head with binary cross entropy loss $\ell_{\text{pred}} = \sum_i^k \lambda_i \mathscr{L}_{\text{CE}}(p_i, y)$

(e) **Slice-aware representation:** For each example, the slice-aware representation is the combinatino of several "expert" slice representations according to 1) the likelihood that the input is in the slice and 2) the confidence of the slice "expert's" prediction. To explicitly model the residual from slice representations to the base representation, we initialize a trivial "base slice" which consists of *all examples* so that we have the corresponding indicators, q_{BASE}, and predictions, p_{BASE}.

Let $Q = \{q_1, \ldots, q_k, q_{\text{BASE}}\} \in \mathbb{R}^{k+1}$ be the vector of concatenated slice indicator logits, $P = \{p_1, \ldots, p_k, p_{\text{BASE}}\} \in \mathbb{R}^{k+1}$ be the vector of concatenated slice prediction logits, and $R = \{r_1, \ldots, r_k, r_{\text{BASE}}\} \in \mathbb{R}^{h \times k+1}$ be the $k+1$ stacked slice expert representations. We compute our attention by combining the likelihood of slice membership, Q, and the slice prediction confidence, which we interpret the absolute value of the binary logits, $abs(P)$. We then apply a Softmax to create soft attention weights over the k+1 slice expert representations: $a \in \mathbb{R}^{k+1} = \text{Softmax}(Q + abs(P))$. Using a weighted sum, we then compute the combined, slice-aware representation: $z' \in \mathbb{R}^h = Ra$

(f) **Prediction head** Finally, we use our slice-aware representation z' as the input to a final linear layer, $f(\cdot)$, which we term the *prediction head*, to make a prediction on the original, base task. During inference, this prediction head makes the final prediction. To train the prediction head, we minimize the cross entropy between the prediction head's output, $f(z')$, and the base task's ground truth labels, y: $\ell_{\text{base}} = \mathscr{L}_{\text{CE}}(f(z'), y)$.

Overall, the model is trained using loss values from all task heads: $\ell_{\text{train}} = \ell_{\text{base}} + \ell_{\text{ind}} + \ell_{\text{pred}}$. In Figure 3, we show ablations of this architecture in a synthetic experiment varying the components that are considered the reweighting mechanism—specifically, our described attention approach outperforms using *only* indicator outputs, *only* predictor confidences, or uniform weights to reweight the slice representations.

3.3 Synthetic data experiments

To understand the properties of *Slice-based Learning*, we validate our model and its components (Figure 2) on a set of synthetic data. In the results demonstrated in Figure 1, we construct a dataset $\mathscr{X} \in \mathbb{R}^2$ with a 2-way classification problem in which over 95% of the data are linearly separable. We introduce two minor perturbations along the decision boundary, which we define as critical slices, $s1$ and $s2$. Intuitively, examples that fall within these slices follow different distributions $\left(P(\mathscr{Y} \mid \mathscr{X}, s_i)\right)$ relative to the overall data $(P(\mathscr{Y} \mid \mathscr{X}))$.

For all models, the shared backbone is defined as a 2-layer MLP architecture with a backbone representation size $d = 13$ and a final $ReLU$ non-linearity. OURSis initialized with a slice-representation size $h = 13$.

The model learns the slice-conditional label distribution $P(Y|s_i,X)$ from noisy SF inputs.—We show in Figure 1b that the slices at the perturbed decision boundary cannot be learned in the general case, by a VANILLA model. As a result, we define two SFs, λ_1 and λ_2, to target the slices of interest. Because our attention-based model (OURS) is slice-aware, it outperforms VANILLA, which has no notion of slices (Figure 4, left). Intuitively, if the model knows "where" in the 2-dim data space an example lives (as defined by SFs), it can condition on slice information as it makes a final prediction. In Figure 5, we observe our model's ability to cope with noisy SF inputs: the indicator is robust to moderate amounts of noise by ignoring noisy examples (middle); with extremely noisy inputs, it disregards poorly-defined SFs by assigning relatively uniform weights (right).

Overall model performance does not degrade.—The primary goal of the slice-aware model is to improve slice-specific performance without degrading the model's existing capabilities. We show that OURS improves the overall score by 1.36 F1 points by learning the proportionally smaller perturbations in the decision boundary in addition to the more general linear boundary (Figure 4, left). Further, we note that we do not regress on individual slice heads.

Learning slice weights with features $P(Y|s_i,X)$ improves over doing so with only supervision source information $P(Y|s_i)$.—A core assumption of our approach asserts that if the model learns improved slice-conditional weights via λ_i, downstream slice-specific performance will improve. Data programming (DP) [28] is a popular weak supervision approach deployed at numerous Fortune 500 companies [2], in which the weights of heuristics are learned solely from labeling source information. We emphasize that our setting provides the model with strictly more information—in the data's feature representations—to learn such weights; we show in Figure 4 (right) that increasing representation size allows us to significantly outperform DP.

Attention weights learn from noisy λ_i to combine slice residual representations.—Given slice information, the model achieves improvements over methods that do not aggregate slice information, as defined by each noisy λ_i. Both the indicator outputs (Q) and prediction confidence ($abs(P)$) are robustly combined in the attention mechanism. Even a noisy indicator will be upweighted if the predictions are high confidence, and if the indicator has high signal, even a slice expert making poor predictions can benefit from the underlying features. We show in Figure 4 that our method improves over HPS, which is slice-aware, but has no way of combining slice information despite increasingly noisy λ_i. In contrast, our attention-based architecture is able to combine slice expert representations, and (OURS) sees improvements over VANILLA of 38.2 slice-level F1 averaged across S_1 and S_2.

Our model demonstrates similar expressivity to MoE with much less cost.—We come within 6.25 slice-level F1 averaged across S_1 and S_2 of MoE with approximately half as many parameters (Figure 4). With large backbone architectures, characterized by M parameters, and a large number of slices, k, MoE requires a quadratically large number of

parameters, because we initialize an entire backbone for each slice. In contrast, all other models scale linearly in parameters with M.

4 Experiments

In several text and image-based applications, we demonstrate that using the same backbone architecture as baselines, our approach succesfully models slice importance and significantly improves slice-level performance without impacting overall model performance. Then, we demonstrate our method's advantages in aggregating noisy heuristics, compared to existing weak supervision literature.

4.1 Applications

We compare our method to several baselines that capture alternatives we have seen in practice or the literature and on natural language understanding (NLU) and computer vision (CV) datasets.

4.1.1 Baselines—

For each baseline, we first train the backbone parameters with a standard hyperparameter search over learning rate and ℓ_2 regularization values. Then, each method is initialized from the backbone weights and fine-tuned for a fixed number of epochs and the optimal hyperparameters. We perform all empirical experiments on Google's Cloud infrastructure using NVIDIA V100 GPUs.

VANILLA: A vanilla neural network backbone is trained with a final prediction head to make predictions. This baseline represents the de-facto approach used in deep learning modeling tasks; it is unaware of the notion of slices and, as a result, neglects to model them.

MoE: We train a *mixture of experts* [17], where each *expert* is a separate VANILLA model trained on a data subset specified by the SF, λ_i. A gating network [33] is then trained to combine expert predictions into a final prediction.

HPS: In the style of multi-task learning, we model slices as separate task heads with a shared backbone trained via *hard parameter sharing*. Each slice task performs the same prediction task, but they are trained on subsets of data corresponding to λ_i. In this approach, backpropagation from different slice tasks is intended to encourage a slice-aware representation bias [7, 31].

MANUAL: To simulate the manual effort required to upweight hyperparameters for tuning slice-specific representations, we leverage the same architecture as HPS and grid search over multipliers for loss terms, $\alpha \in \{2,20,50,100\}$, of underperforming slices (i.e. where $\text{score}_{\text{overall}} - \text{score}_{\text{slice}} \geq 5$ F1 points in VANILLA).

4.1.2 Datasets

NLU Datasets.: We select slices based on independently-conducted error analyses [19] (Appendix A1.2). In **Corpus of Linguistic Acceptability (COLA)** [36], the task is to predict whether a sentence is linguistically acceptable (i.e. grammatically); we measure

performance using the Matthews correlation coefficient [23]. Natural slices might occur as questions or long sentences, as corresponding examples might consist of non-standard or challenging sentence structure. Since ground truth test labels are not available for this task (they are held out in evaluation servers [35]), we sample to create data splits with 7.2K/1.3K/1K train/valid/test sentences, respectively. To properly evaluate slices of interest, we ensure that the proportions of examples in ground truth slices are consistent across splits. In **Recognizing Textual Entailment (RTE)** [3, 4, 10, 14, 35], the task is to predict whether or not a premise sentence entails a hypothesis sentence. Similar to COLA, we create our own data splits and use 2.25K/0.25K/0.275K train/valid/test sentences, respectively. Finally, in a user study where we work with practitioners tackling the **SuperGlue** [34] benchmark, we leverage *Slice-based Learning* to improve state-of-the-art model quality on benchmark submissions.

CV Dataset.: In the image domain, we evaluate on an autonomous vehicle dataset called **Cyclist Detection for Autonomous Vehicles (CYDET)** [21]. We leverage clips in a self-driving video dataset to detect whether a cyclist (person plus bicycle) is present at each frame. We select one independent clip for evaluation, and the remainder for training; for valid/test splits, we select alternating batches of five frames each from the evaluation clip. We preprocess the dataset with an open-source implementation of Mask R-CNN [22] to provide metadata (e.g. presence of traffic lights, benches), which serve as slice indicators for each frame.

4.1.3 Results

Slice-aware models improve slice-specific performance.: We see in Table 1 that each slice-aware model (HPS, MANUAL, MoE, OURS) largely improves over the naive model.

OURS improves overall performance.: We also observe that OURS improves overall performance for each of the datasets. This is likely because the chosen slices were explicitly modeled from error analysis papers, and explicitly modeling “error” slices led to improved overall performance.

OURS learns slice expert representations consistently.: While HPS and MANUAL perform well on some slices, they exhibit much higher variance compared to OURS and MoE (as denoted by the std. in Table 1). These baselines lack an attention mechanism to reweight slice representations in a consistent way; instead, they rely purely on representation bias from slice-specific heads to improve slice-level performance. Because these representations are not modeled explicitly, improvements are largely driven by chance, and this approach risks worsening performance on other slices or overall.

OURS improves performance with a parameter-efficient representation.: For **CoLA** and **RTE** experiments, we used the BERT-base [12] architecture with 110M parameters; for **CyDet**, we used ResNet-18 [15]. For each additional slice, OURS requires a 7% and 5% increase in relative parameter count in the BERT and ResNet architectures, respectively (total relative parameter increase reported in Table 1). As a comparison, HPS requires the same relative increase in parameters per slice. MoE on the other hand, increases relative

number of parameters by 100% for both architectures. With limited increase in model size, OURS outperforms or matches all other baselines, including MoE, which requires an order of magnitude more parameters.

OURS improves state-of-the-art quality models with slice-aware representations.: In a submission to SuperGLUE benchmark evaluation servers, we leverage the same BERT-large architecture of previous submissions and observe improvements of +3.8/+2.8 avg. F1/acc. on CB, +2.4 acc. on COPA, +2.5 acc. on WiC; this amounts to an aggregate 2.7 point increase in overall benchmark score.

4.2 Weak Supervision Comparisons

To contextualize our contributions in the weak supervision literature, we compare directly to Data Programming (DP) [26], a popular approach for reweighting user-specified heuristics using supervision source information [28]. We consider two text-based relation extraction datasets: **Chemical-Disease Relations (CDR)**,[37], in which we identify causal links between chemical and disease entities in a dataset of PubMed abstracts, and **Spouses** [9], in which we identify mentions of spousal relationships using preprocessed pairs of person mentions from news articles (via Spacy [16]). In both datasets, we leverage the exact noisy linguistic patterns and distant supervision heuristics provided in the opensource implementation of DP. Rather than voting on a particular class, we repurpose the provided labeling functions as binary slice indicators for our model. We then train our slice-aware model on the probabilistic labels aggregated from these heuristics.

OURS improves over current weak supervision methods.: Treating the noisy heuristics as slicing functions, we observe lifts of up to 1.3 F1 overall and 15.9 F1 on heuristically-defined slices. We reproduce the DP [26] setup to obtain overall scores of F1=41.9 on **Spouses** and F1=56.4 on **CDR**. Using *Slice-based Learning*, we improve to 42.8 (+0.9) and 57.7 (+1.3) F1, respectively. Intuitively, we can explain this improvement, because OURS has access to features of the data belonging to slices whereas DP relies only on the source information of each heuristic.

5 Conclusion

We introduced the challenge of improving slice-specific performance without damaging the overall model quality, and introduced the first programming abstraction and machine learning model to support these actions. We demonstrated that the model could be used to push the state-of-the-art quality. In our analysis, we can explain consistent gains in the *Slice-based Learning* paradigm because our attention mechanism has access to a rich set of deep features, whereas existing weak supervision paradigms have no way to access this information. We view this work in the context of programming models that sit on top of traditional modeling approaches in machine learning systems.

Acknowledgements

We would like to thank Braden Hancock, Feng Niu, and Charles Srisuwananukorn for many helpful discussions, tests, and collaborations throughout the development of slicing. We gratefully acknowledge the support of DARPA

under Nos. FA87501720095 (D3M), FA86501827865 (SDH), FA86501827882 (ASED), NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity) and CCF1563078 (Volume to Velocity), ONR under No. N000141712266 (Unifying Weak Supervision), the Moore Foundation, NXP, Xilinx, LETI-CEA, Intel, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, and American Family Insurance, Google Cloud, Swiss Re, and members of the Stanford DAWN project: Teradata, Facebook, Google, Ant Financial, NEC, SAP, VMWare, and Infosys. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NIH, ONR, or the U.S. Government.

A1: Appendix

A1.1 Model Characteristics

We include summarized model characteristics and the associated baselines to supplement Sections 3.3 and 4.1.1.

A1.2 Slicing Function (SF) Construction

We walk through specific examples of SFs written for a number of our applications.

Textual SFs For text-based applications (COLA, RTE), we write SFs over pairs of sentences for each task. Following dataset convention, we denote the first sentence as the *premise* and the second as the *hypothesis* where appropriate. Then, SFs are written, drawing largely from existing error analysis [19]. For instance, we might expect certain questions to be especially difficult to formulate in a language acceptability task. So, we write the following SF to heuristically target *where* questions:

```
def SF_where_question(premise, hypothesis):
# triggers if “where” appears in sentence
sentences = premise + hypothesis
return “where” in sentences.lower()
```

In some cases, we write SFs over both sentences at once. For instance, to capture possible errors in article references (e.g. *the Big Apple* vs *a big apple*), we specify a slice where multiple instances of the same article appear in provided sentences:

```
def SF_has_multiple_articles(premise, hypothesis):
# triggers if a sentence has more than one occurrence of the same article
sentences = premise + hypothesis
multiple_a = sum([int(x == “a”) for x in sentences.split()]) > 1
multiple_an = sum([int(x == “an”) for x in sentences.split()]) > 1
multiple_the = sum([int(x == “the”) for x in sentences.split()]) > 1
return multiple_a or multiple_an or multiple_the
```

Image-based SFs For computer vision applications, we leverage image metadata and bounding box attributes, generated from an off-the-shelf Mask R-CNN [22], to target slices of interest.

```
def SF_bus(image):
# triggers if a “bus” appears in the predictions of the noisy detector
outputs = noisy_detector(image)
return “bus” in outputs
```

We note that these potentially expensive detectors are *non-servable*—they run offline, and our model uses learned indicators at inference time. Despite the detectors’ noisy predictions, our model is able to to reweight representations appropriately.

A1.3 COLA SFs

COLA is a language acceptability task based on linguistics and grammar for individual sentences. We draw from error analysis which introduces several linguistically imortance slices for language acceptability via a series of challenge tasks. Each task consists of synthetically generated examples to measure model evaluation on specific slices. We heuristically define SFs to target subsets of data corresponding to each challenge, and include the full set of SFs derived from each category of challenge tasks:

- **Wh-words:** This task targets sentences containing *who, what, where, when, why, how*. We exclude *why* and *how* below because the COLA dataset does not have enough examples for proper training and evaluation of these slices.

```
def SF_where_in_sentence(sentence):
return “where” in sentence
def SF_who_in_sentence(sentence):
return “who” in sentence
def SF_what_in_sentence(sentence):
return “what” in sentence
def SF_when_in_sentence(sentence):
return “when” in sentence
```

- **Definite-Indefinite Articles:** This challenge measures the model based on different combinations of definite (*the*) and indefinite (*a,an*) articles in a sentence (i.e. swapping definite for indefinite articles and vice versa). We target containing multiple uses of a definite (*the*) or indefinite article (*a, an*):

```
def SF_has_multiple_articles(sentence):
# triggers if a sentence has more than one occurrence of the same article
multiple_indefinite = sum([int(x == “a”) for x in sentence.split()]) > 1 or
```

```
sum([
int(x == “an”) for x in sentence.split()]) > 1
multiple_definite = sum([int(x == “the”) for x in sentence.split()]) > 1
return multiple_indefinite or multiple_definite
```

- **Coordinating Conjunctions:** This task seeks to measure correct usage of coordinating conjunctions (*and, but, or*) in context. We target the presence of these words in both sentences.

```
def and_in_sentence(sentence):
return “and” in sentence
def but_in_sentence(sentence):
return “but” in sentence
def or_in_sentence(sentence):
return “or” in sentence
```

- **End-of-Sentence:** This challenge task measures a model’s ability to identify coherent sentences or sentence chunks after removing puctuation. We heuristically target this slice by identifying particularly short sentences and those that end with verbs and adverbs. We use off-the-shelf parsers (i.e. Spacy [16]) to generate part-of-speech tags.

```
def SF_short_sentence(sentence):
# triggered if sentence has fewer than 5 tokens
return len(sentence.split()) < 5
# Spacy tagger
def get_spacy_pos(sentence):
import spacy
nlp = spacy.load(“en_core_web_sm”)
return nlp(sentence).pos_
def SF_ends_with_verb(sentence):
# remove last token, which is always is punctuation
sentence = sentence[:-1]
return get_spacy_pos(sentence)[-1] == “VERB”
def SF_ends_with_adverb(sentence):
# remove last token, which is always is punctuation
sentence = sentence[:-1]
return get_spacy_pos(sentence)[-1] == “ADVERB”
```

A1.4 RTE SFs

Similar to COLA, we use challenge tasks from NLI-based error analysis [19] to write SFs over the textual entailment (RTE) dataset.

- **Prepositions:** In one challenge, the authors swap prepositions in the dataset with prepositions in a manuallycurated list. The list in its entirety spans a large proportion of the RTE dataset, which would constitute a very large slice. We find it more effective to separate these prepositions into *temporal* and *possessive* slices.

```
def SF_has_temporal_preposition(premise, hypothesis):
temporal_prepositions = [“after”, “before”, “past”]
sentence = premise + sentence
return any([p in sentence for p in temporal_prepositions])
def SF_has_possessive_preposition(premise, hypothesis):
possessive_prepositions = [“inside of”, “with”, “within”]
sentence = premise + sentence
return any([p in sentence for p in possessive_prepositions])
```

- **Comparatives:** One challenge chooses sentences with specific comparative words and mutates/negates them.

We directly target keywords identified in their approach.

```
def SF_is_comparative(premise, hypothesis):
comparative_words = [“more”, “less”, “better”, “worse”, “bigger”, “smaller”]
sentence = premise + hypothesis
return any([p in sentence for p in comparative_words])
```

- **Quantification:** One challenge tests natural language understanding with common quantifiers. We target common quantifiers in both the combined premise/hypothesis and in *only* the hypothesis.

```
def is_quantification(premise, hypothesis):
quantifiers = [“all”, “some”, “none”]
sentence = premise + hypothesis
return any([p in sentence for p in quantifiers])
def is_quantification_hypothesis(premise, hypothesis):
quantifiers = [“all”, “some”, “none”]
return any([p in hypothesis for p in quantifiers])
```

- **Spatial Expressions:** This challenge identifies spatial relations between entities (i.e. *A is to the left of B*). We exclude this task from our slices, because such slices do not account for enough examples in the *RTE* dataset.
- **Negation**: This challenge task identifies whether natural language inference models can handle negations. We heuristically target this slice via a list of common negation words from a top result in a web search.

```
def SF_common_negation(premise, hypothesis):
# Words from https://www.grammarly.com/blog/negatives/
negation_words = [
“no”,
“not”,
“none”,
“no one”,
“nobody”,
“nothing”,
“neither”,
“nowhere”,
“never”,
“hardly”,
“scarcely”,
“barely”,
“doesnt”,
“isnt”,
“wasnt”,
“shouldnt”,
“wouldnt”,
“couldnt”,
“wont”,
“cant”,
“dont”,
]
sentence = premise + hypothesis
return any([x in negation_words for x in sentence])
```

- **Premise/Hypothesis Length**: Finally, separate from the cited error analysis, we target different length hypotheses and premises as an additional set of slicing tasks. In our own error analysis of the RTE model, we found these represented intuitive slices: long premises are typically harder to parse for key information, and shorter hypotheses tend to share syntactical structure.

```
def SF_short_hypothesis(premise, hypothesis):
return len(hypothesis.split()) < 5
def SF_long_hypothesis(premise, hypothesis):
return len(hypothesis.split()) > 100
def SF_short_premise(premise, hypothesis):
return len(premise.split()) < 15
def SF_long_premise(premise, hypothesis):
return len(premise.split()) > 100
```

A1.5 CYDET SFs

For the cyclist detection dataset, we identify subsets that correspond to other objects in the scene using a noisy detector (i.e. an off-the-shelf Mask R-CNN [22]).

```
# define noisy detector
def noise_detector(image):
probs = mask_rcnn.forward(image)
# threshold predictions
preds = []
for object in classes:
if probs[“object”] > 0.5:
preds.append(object)
return preds
# Cyclist Detection SFs
def SF_bench(image):
outputs = noisy_detector(image)
return “bench” in outputs
def SF_truck(image):
outputs = noisy_detector(image)
return “truck” in outputs
def SF_car(image):
outputs = noisy_detector(image)
return “car” in outputs
def SF_bus(image):
outputs = noisy_detector(image)
return “bus” in outputs
def SF_person(image):
outputs = noisy_detector(image)
return “person” in outputs
def SF_traffic_light(image):
outputs = noisy_detector(image)
return “traffic light” in outputs
def SF_fire_hydrant(image):
outputs = noisy_detector(image)
return “fire hydrant” in outputs
def SF_stop_sign(image):
outputs = noisy_detector(image)
return “stop sign” in outputs
def SF_bicycle(image):
outputs = noisy_detector(image)
return “bicycle” in outputs
```

A1.6 Slice-specific Metrics

We visualize slice-specific metrics across each application dataset, for each method of comparison. We report the corresponding aggregate metrics in Figure 1 (below).

In COLA, we see that MoE and OURS exhibit the largest slice-specific gains, and also overfit on the same slice *ends with adverb*. In RTE, we see that OURS improves performance on all slices except *common negation*, where it falls less than a point below VANILLA. On CYDET, we see the largest gains for OURS on *bench* and *bus* slices—in particular, we are able to improve in cases where the model might able to use the presence of these objects to make more informed decisions about whether a cyclist is present. Note: because the MoE model on CYDET encounters an "Out of Memory" error, the corresponding (blue) data bar is not available for this dataset.

References

[1]. Stephen H Bach Daniel Rodriguez, Liu Yintao, Luo Chong, Shao Haidong, Xia Cassandra, Sen Souvik, Ratner Alex, Hancock Braden, Alborzi Houman, et al. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In Proceedings of the 2019 International Conference on Management of Data, pages 362–375. ACM, 2019.

[2]. Stephen H Bach Daniel Rodriguez, Liu Yintao, Luo Chong, Shao Haidong, Xia Cassandra, Sen Souvik, Ratner Alexander, Hancock Braden, Alborzi Houman, et al. Snorkel drybell: A case study in deploying weak supervision at industrial scale. arXiv preprint arXiv:1812.00417, 2018.

[3]. Roy Bar-Haim Ido Dagan, Dolan Bill, Ferro Lisa, Giampiccolo Danilo, Magnini Bernardo, and Szpektor Idan. The second pascal recognising textual entailment challenge. In Proceedings of the second PASCAL challenges workshop on recognising textual entailment, volume 6, pages 6–4. Venice, 2006.

[4]. Bentivogli Luisa, Clark Peter, Dagan Ido, and Giampiccolo Danilo. The fifth pascal recognizing textual entailment challenge. In TAC, 2009.

[5]. Berend Daniel and Kontorovich Aryeh. Consistency of weighted majority votes. In Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS'14, pages 3446–3454, Cambridge, MA, USA, 2014 MIT Press.

[6]. Blum Avrim and Mitchell Tom. Combining labeled and unlabeled data with co-training. In Proceedings of the eleventh annual conference on Computational learning theory, pages 92–100. ACM, 1998.

[7]. Caruana Rich. Multitask learning. Machine learning, 28(1):41–75, 1997.

[8]. Cheng Hao, Fang Hao, and Ostendorf Mari. Open-domain name error detection using a multi-task rnn. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 737–746, 2015.

[9]. Corney David, Albakour Dyaa, Miguel Martinez-Alvarez, and Samir Moussa. What do a million news articles look like? In NewsIR@ ECIR, pages 42–47, 2016.

[10]. Dagan Ido, Glickman Oren, and Magnini Bernardo. The pascal recognising textual entailment challenge In Machine Learning Challenges Workshop, pages 177–190. Springer, 2005.

[11]. Dalvi Nilesh, Dasgupta Anirban, Kumar Ravi, and Rastogi Vibhor. Aggregating crowdsourced binary ratings. In Proceedings of the 22Nd International Conference on World Wide Web, WWW '13, pages 285–294, New York, NY, USA, 2013 ACM.

[12]. Devlin Jacob, Chang Ming-Wei, Lee Kenton, and Toutanova Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[13]. Mitchell Tet al. Never-ending learning. In AAAI, 2015.

[14]. Giampiccolo Danilo, Magnini Bernardo, Dagan Ido, and Dolan Bill. The third pascal recognizing textual entailment challenge. In Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing, pages 1–9. Association for Computational Linguistics, 2007.

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

[16]. Honnibal Matthew and Johnson Mark. An improved non-monotonic transition system for dependency parsing. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1373–1378, Lisbon, Portugal, 9 2015 Association for Computational Linguistics.

[17]. Jacobs Robert A, Jordan Michael I, Nowlan Steven J, and Hinton Geoffrey E. Adaptive mixtures of local experts. Neural computation, 3(1):79–87, 1991. [PubMed: 31141872]

[18]. Karpathy Andrej. Building the software 2.0 stack, 2019.

[19]. Kim Najoung, Patel Roma, Poliak Adam, Wang Alex, Xia Patrick, R Thomas McCoy Ian Tenney, Ross Alexis, Linzen Tal, Van Durme Benjamin, et al. Probing what different nlp tasks teach machines about function word comprehension. arXiv preprint arXiv:1904.11544, 2019.

[20]. Mann GS and McCallum A. Generalized expectation criteria for semi-supervised learning with weakly labeled data. JMLR, 11(Feb):955–984, 2010.

[21]. Masalov Alexerand, Ota Jeffrey, Corbet Heath, Lee Eric, and Pelley Adam. Cydet: Improving camerabased cyclist recognition accuracy with known cycling jersey patterns. In 2018 IEEE Intelligent Vehicles Symposium (IV), pages 2143–2149. IEEE, 2018.

[22]. Massa Francisco and Girshick Ross. maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark, 2018 Accessed: Feb 2019.

[23]. Matthews Brian W. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica Acta (BBA)-Protein Structure, 405(2):442–451, 1975.

[24]. Mintz M, Bills S, Snow R, and Jurafsky D. Distant supervision for relation extraction without labeled data. In Proc ACL, pages 1003–1011, 2009.

[25]. Misra Ishan, Shrivastava Abhinav, Gupta Abhinav, and Hebert Martial. Cross-stitch networks for multi-task learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3994–4003, 2016.

[26]. Ratner AJ, Bach SH, Ehrenberg H, Fries J, Wu S, and Ré C. Snorkel: Rapid training data creation with weak supervision. In VLDB, 2018.

[27]. Ratner Alexander, Hancock Braden, and Christopher Ré. The role of massively multi-task and weak supervision in software 2.0. 2019.

[28]. Ratner Alexander J, De Sa Christopher M, Wu Sen, Selsam Daniel, and Ré Christopher. Data programming: Creating large training sets, quickly. In Advances in neural information processing systems, pages 3567–3575, 2016. [PubMed: 29872252]

[29]. Christopher Ré Feng Niu, Gudipati Pallavi, and Srisuwananukorn Charles. Overton: A data system for monitoring and improving machine-learned products. 2019.

[30]. Rei Marek. Semi-supervised multitask learning for sequence labeling. arXiv preprint arXiv:1704.07156, 2017.

[31]. Ruder Sebastian. An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098, 2017.

[32]. Sagi Omer and Rokach Lior. Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8(4):e1249, 2018.

[33]. Sigaud Olivier, Masson Clément, Filliat David, and Stulp Freek. Gated networks: an inventory. arXiv preprint arXiv:1512.03201, 2015.

[34]. Wang Alex, Pruksachatkun Yada, Nangia Nikita, Singh Amanpreet, Michael Julian, Hill Felix, Levy Omer, and Bowman Samuel R. Superglue: A stickier benchmark for general-purpose language understanding systems. arXiv preprint arXiv:1905.00537, 2019.

[35]. Wang Alex, Singh Amapreet, Michael Julian, Hill Felix, Levy Omer, and Samuel R Bowman. Glue: A multitask benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461, 2018.

[36]. Warstadt Alex, Singh Amanpreet, and Samuel R Bowman. Neural network acceptability judgments. arXiv preprint arXiv:1805.12471, 2018.

[37]. Wei Chih-Hsuan, Peng Yifan, Leaman Robert, Allan Peter Davis, Carolyn J Mattingly, Jiao Li, Thomas C Wiegers, and Zhiyong Lu. Overview of the biocreative v chemical disease relation (cdr) task. In Proceedings of the fifth BioCreative challenge evaluation workshop, pages 154–166, 2015.

[38]. Yang Yongxin and Hospedales Timothy. Deep multi-task representation learning: A tensor factorisation approach. arXiv preprint arXiv:1605.06391, 2016.

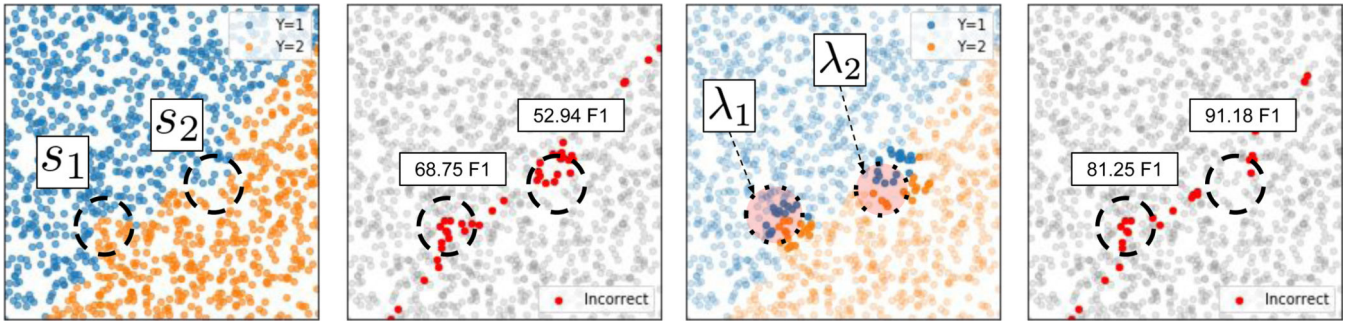

Figure 1: *Slice-based Learning* via synthetically generated data:
(a) The data distribution contains critical slices, s_1, s_2, that represent a small proportion of the dataset. (b) A vanilla neural network correctly learns the general, linear decision boundary but fails to learn the perturbed slice boundary. (c) A user writes *slicing functions* (SFs), λ_1, λ_2, to heuristically target critical subsets. (d) The model commits additional capacity to learn slice expert representations. Upon reweighting slice expert representations, the slice-aware model learns to classify the fine-grained slices with higher F1 score.

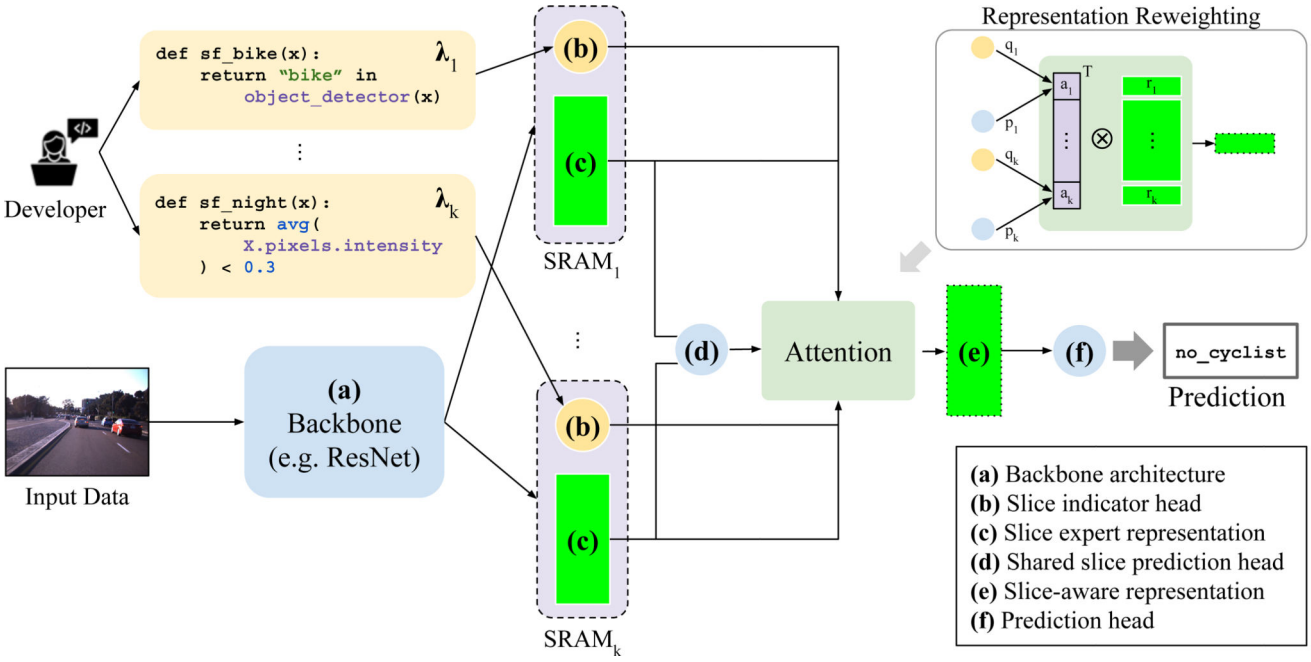

Figure 2: Model Architecture:

A developer writes SFs ($\lambda_{i=1,\ldots,k}$) over input data and specifies any. **(a) backbone** architecture (e.g. ResNet [15], BERT [12]) as a feature extractor. Extracted features are shared parameters for k slice-residual attention modules; each learns a **(b) slice indicator head**, which is supervised by a corresponding λ_i, and a **(c) slice expert representation**, which is trained only on examples belonging to the slice using a **(d) shared slice prediction head**. An attention mechanism reweights these representations into a combined, **(e) slice-aware representation**. A final **(f) prediction head** makes model predictions based on the slice-aware representation.

Reweighting Mechanism	Performance (F1 score)			
	Overall	S_0	S_1	S_2
UNIFORM	77.1	57.1	68.6	73.6
IND. OUTPUT	78.1	52.6	71.0	76.4
PRED. CONF.	79.3	61.1	69.2	78.7
FULL ATTENTION (Ind. Output + Pred. Conf.)	**82.7**	**66.7**	**77.4**	**89.1**

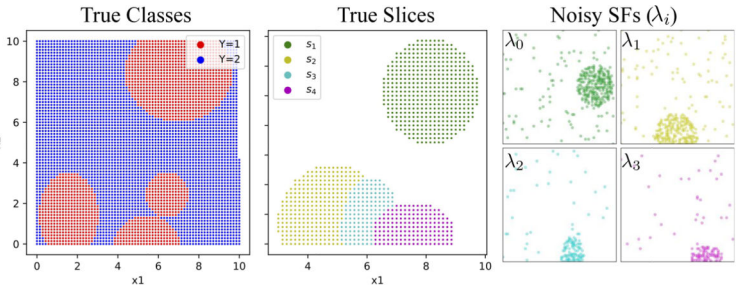

Figure 3: Architecture Ablation:

Using a synthetic, two-class dataset (Figure, left) with four randomly specified (size, shape, location) slices (Figure, middle), we specify corresponding, noisy SFs (Figure, right) and ablate specific model components by modifying the reweighting mechanism for slice expert representations. We compare overall/slice performance for uniform, indicator output, prediction confidence weighting, and the proposed attention weighting using all components. Our FULL ATTENTION approach performs most consistently on slices without worsening overall performance.

Method	Performance (F1 score) Overall	S_1	S_2
VANILLA	96.56	52.94	68.75
DP [28]	96.88	44.12	43.75
HPS [7]	96.72	50.00	75.00
MOE [17]	**98.48**	88.24	**87.50**
OURS	97.92	**91.18**	81.25

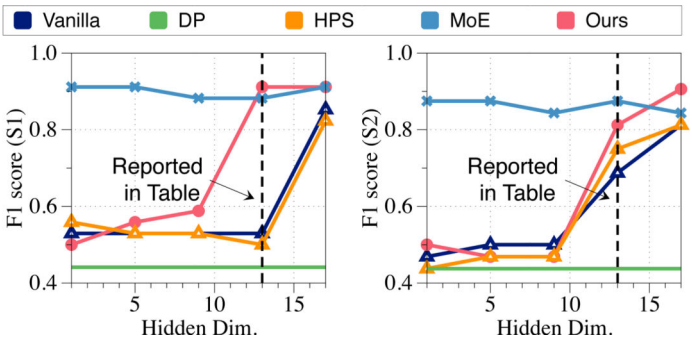

Figure 4: Scaling with hidden feature representation dimensions.

We plot model quality versus the hidden dimension size. The slice-aware model (OURS) improves over *hard parameter sharing* (HPS) on both slices at a fixed hidden dimension size, while being close to *mixture of experts* (MoE).

Note: MoE has significantly more parameters overall, as it copies the entire model.

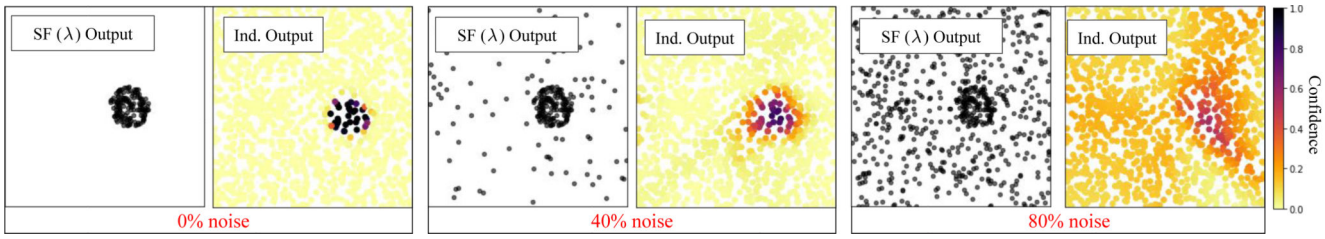

Figure 5: Coping with Noise:

We test the robustness of our approach on a simple synthetic example. In each panel, we show noisy SFs (left) as binary points and the corresponding slice indicator's output (right) as a heatmap of probabilities. We show that the indicator assigns low relative probabilities on noisy (40%, middle) samples and ignores a very noisy (80%, right) SF, assigning relatively uniform scores to all samples.

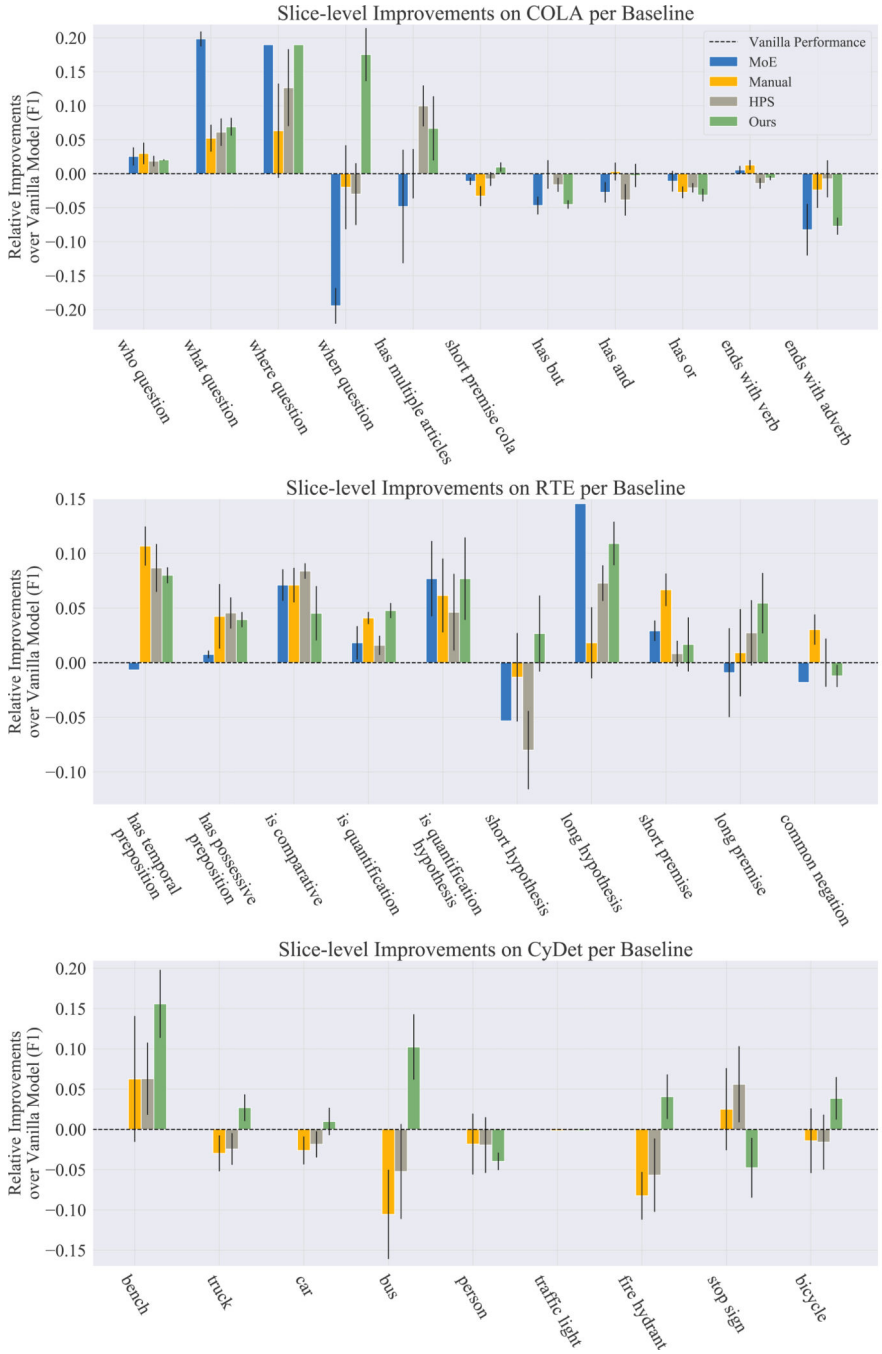

Figure 6:

For each application dataset (Section 4.1) we report all relative, slice-level metrics compared to VANILLA for each model.

Table 1:

Application Datasets:

We compare our model to baselines averaged over 5 runs with different seeds in natural language understanding and computer vision applications and note the relative increase in number of params for each method. We report the overall score and maximum relative improvement (i.e. Lift) over the VANILLA model for each of the slice-aware baselines. For some trials of MoE, our system ran out of GPU memory (i.e. OOM).

Dataset	**COLA (Matthews Corr. [23])**				**RTE (F1 Score)**				**CYDET (F1 Score)**			
	Param Inc.	**Overall (std)**	**Slice Lift**		**Param Inc.**	**Overall (std)**	**Slice Lift**		**Param Inc.**	**Overall (std)**	**Slice Lift**	
			Max	**Avg**			**Max**	**Avg**			**Max**	**Avg**
VANILLA	-	57.8 (±1.3)	-	-	-	67.0 (±1.6)	-	-	-	39.4 (±-5.4)	-	-
HPS [7]	**12%**	57.4 (±2.1)	+12.7	1.1	**10%**	67.9 (±1.8)	**+12.7**	+2.9	**10%**	37.4 (±3.6)	+6.3	−0.7
MANUAL	**12%**	57.9 (±1.2)	+6.3	+0.4	**10%**	69.4 (±1.8)	+10.7	+4.2	**10%**	36.9 (±4.2)	+6.3	−1.7
MoE [17]	100%	57.2 (±0.9)	**+20.0**	+1.3	100%	69.2 (±1.5)	+10.9	+3.9	100%	OOM	OOM	OOM
Ours	**12%**	**58.3** (±0.7)	+19.0	**+2.5**	**10%**	**69.5** (±0.8)	+10.9	**+4.6**	**10%**	**40.9** (±3.9)	**+15.6**	**+2.3**

Table 2:

Model characterizations:

We characterize each model's advantages/limitations and compute the number of parameters for each baseline model, given k slices, M backbone parameters, feature representation z dimension r, and slice expert representation p_i dimension h.

Method	Slice-aware	No manual tuning	Weighted slice info.	Avoids copies of model (M params)	Num. Params
VANILLA		✓		✓	O(M+r)
HPS	✓	✓		✓	O(M+kr)
MANUAL	✓		✓	✓	O(M+kr)
MoE	✓	✓	✓		O(kM+kr)
OURS	✓	✓	✓	✓	O(M+krh)