Learning to Classify Intents and Slot Labels Given a Handful of Examples

Jason Krone
Amazon AI
Palo Alto, CA
kronej@amazon.com

Yi Zhang
Amazon AI
Seattle, WA
yizhngn@amazon.com

Mona Diab[*]
The George Washington University
Washington, DC
mtdiab@gwu.edu

Abstract

Intent classification (IC) and slot filling (SF) are core components in most goal-oriented dialogue systems. Current IC/SF models perform poorly when the number of training examples per class is small. We propose a new few-shot learning task, *few-shot IC/SF*, to study and improve the performance of IC and SF models on classes not seen at training time in ultra low resource scenarios. We establish a *few-shot IC/SF* benchmark by defining few-shot splits for three public IC/SF datasets, ATIS, TOP, and SNIPS. We show that two popular few-shot learning algorithms, model agnostic meta learning (MAML) and prototypical networks, outperform a fine-tuning baseline on this benchmark. Prototypical networks achieves substantial gains in IC performance on the ATIS and TOP datasets, while both prototypical networks and MAML outperform the baseline with respect to SF on all three datasets. In addition, we demonstrate that joint training as well as the use of pre-trained language models, ELMo and BERT in our case, are complementary to these few-shot learning methods and yield further gains.

1 Introduction

In the context of goal-oriented dialogue systems, intent classification (IC) is the process of classifying a user's utterance into an intent, such as *Book-Flight* or *AddToPlaylist*, referring to the user's goal. While slot filling (SF) is the process of identifying and classifying certain tokens in the utterance into their corresponding labels, in a manner akin to named entity recognition (NER). However, in contrast to NER, typical slots are particular to the domain of the dialogue, such as music or travel. As a reference point, we list intent and slot label annotations for an example utterance from the SNIPS dataset with the *AddToPlaylist* IC in Figure 1.

Token	Slot Label
Please	O
add	O
some	O
Pete	AddToPlaylist:artist
Townshend	AddToPlaylist:artist
to	O
my	AddToPlaylist:playlist_owner
playlist	O
Fiesta	AddToPlaylist:playlist
Hits	AddToPlaylist:playlist
con	AddToPlaylist:playlist
Lali	AddToPlaylist:playlist

Figure 1: Tokens and corresponding slot labels for an utterance from the AddToPlaylist intent class in the SNIPS dataset prefixed by intent class name.

As of late, most state-of-the-art IC/SF models are based on feed-forward, convolutional, or recurrent neural networks (Hakkani-Tür et al., 2016; Goo et al., 2018; Gupta et al., 2019). These neural models offer substantial gains in performance, but they often require a large number of labeled examples (on the order of hundreds) per intent class and slot-label to achieve these gains. The relative scarcity of large-scale datasets annotated with intents and slots prohibits the use of neural IC/SF models in many promising domains, such as medical consultation, where it is difficult to obtain large quantities of annotated dialogues.

Accordingly, we propose the task of *few-shot IC/SF*, catering to domain adaptation in low resource scenarios, where there are only a handful of annotated examples available per intent and slot in the target domain. To the best of our knowledge, this work is the first to apply the few-shot learning framework to a joint sentence classification and sequence labeling task. In the NLP literature, few-shot learning often refers to a low resource, cross lingual setting where there is limited data available in the target language. We emphasize that our definition of *few-shot IC/SF* is distinct in that we limit

---

[*]Work performed while at Amazon AI

the amount of data available per target class rather than target language.

Few-shot IC/SF builds on a large body of existing few-shot classification work. Drawing inspiration from computer vision, we experiment with two prominent few shot image classification approaches, *prototypical networks* and *model agnostic meta learning* (MAML). Both these methods seek to decrease over-fitting and improve generalization on small datasets, albeit via different mechanisms. *Prototypical networks* learns class specific representations, called *prototypes*, and performs inference by assigning the class label associated with the prototype closest to an input embedding. Whereas MAML modifies the learning objective to optimize for pre-training representations that transfer well when fine-tuned on a small number of labeled examples.

For benchmarking purposes, we establish few-shot splits for three publicly available IC/SF datasets: ATIS (Hemphill et al., 1990), SNIPS (Coucke et al., 2018), and TOP (Gupta et al., 2018). Empirically, prototypical networks yields substantial improvements on this benchmark over the popular *"fine-tuning"* approach (Goyal et al., 2018; Schuster et al., 2018), where representations are pre-trained on a large, *"source"* dataset and then fine-tuned on a smaller, *"target"* dataset. Despite performing worse on intent classification, MAML also achieves gains over *"fine-tuning"* on the slot filling task. Orthogonally, we experiment with the use of two pre-trained language models, BERT and ELMO, as well as joint training on multiple datasets. These experiments show that the use of pre-trained, contextual representations is complementary to both methods. While prototypical networks is uniquely able to leverage joint training to consistently boost slot filling performance.

In summary, our primary contributions are fourfold:

1. Formulating IC/SF as a few-shot learning task;

2. Establishing few-shot splits[1] for the ATIS, SNIPS, and TOP datasets;

3. Showing that MAML and prototypical networks can outperform the popular "fine-tuning" domain adaptation framework;

4. Evaluating the complementary of contextual embeddings and joint training with MAML and prototypical networks.

2 Related Work

2.1 Few-shot Learning

Early adoption of few-shot learning in the field of computer vision has yielded promising results. Neural approaches to few-shot learning in computer vision fall mainly into three categories: *optimization-*, *metric-*, or *memory-based*. *Optimization-based* methods typically learn an initialization or fine-tuning procedure for a neural network. For instance, MAML (Finn et al., 2017) directly optimizes for representations that generalize well to unseen classes given a few labeled examples. Using an LSTM based meta-learner, Ravi and Larochelle (2016) learn both the initialization and the fine-tuning procedure. In contrast, *metric-based* approaches learn an embedding space or distance metric under which examples belonging to the same class have high similarity. Prototypical networks (Snell et al., 2017), siamese neural networks (Koch, 2015), and matching networks (Vinyals et al., 2016) all belong to this category. Alternatively, memory based approaches apply memory modules or recurrent networks with memory, such as a LSTM, to few-shot learning. These approaches include differentiable extensions to k-nearest-neighbors (Kaiser et al., 2017) and applications of the Neural Turing Machines (Graves et al., 2014; Santoro et al., 2016).

2.2 Few-shot Learning for Text Classification

To date, applications of few-shot learning to natural language processing focus primarily on text classification tasks. Yu et al. (2018) identify *"clusters"* of source classification tasks that transfer well to a given target task, and meta learn a linear combination of similarity metrics across *"clusters"*. The source tasks with the highest likelihood of transfer are used to pre-train a convolutional network that is subsequently fine-tuned on the target task. Han et al. (2018) propose *FewRel*, a few-shot relation classification dataset, and use this data to benchmark the performance of few-shot models, such as *prototypical networks* and *SNAIL* (Mishra et al., 2017). *ATAML* (Jiang et al., 2018), one of the few optimization based approaches to few-shot sentence classification, extends MAML to learn task-specific as well as task agnostic representations

---

[1] Few-shot split intent assignments given in section A.1

using feed-forward attention mechanisms. (Dou et al., 2019) show that further pre-training of contextual representations using *optimization-based* methods benefits downstream performance.

2.3 Few-shot Learning for Sequence Labeling

In one of the first works on few-shot sequence labeling, Fritzler et al. (2019) apply prototypical networks to few-shot named entity recognition by training a separate prototypical network for each named entity type. This design choice makes their extension of prototypical networks more restrictive than ours, which trains a single model to classify all sequence tags. (Hou et al., 2019) apply a CRF based approach that learns emission scores using pre-trained, contextualized embeddings to few-shot SF (on SNIPS) and few-shot NER.

3 Task Formulation

3.1 Few-shot Classification

The goal of *few-shot classification* is to adapt a classifier f_ϕ to a set of new classes L not seen at training time, given a *few* labeled examples per class $l \in L$. In this setting, train and test splits are defined by disjoint class label sets L_{train} and L_{test}, respectively. The classes in L_{train} are made available for pre-training and those in L_{test} are held out for low resource adaptation at test time. Few-shot evaluation is done episodically, i.e. over a number of mini adaptation datasets, called episodes. Each episode consists of a *support set* S and a *query set* Q. The *support set* contains k_l labeled examples $S_l = \{(x_l^i, y_l)|i \in (1 \ldots k_l)\}$ per held out class $l \in L$; we define $S = \bigcup_{l \in L} S_l$. Similarly, the *query set* contains k_q labeled instances $Q_l = \{(x_l^j, y_l)|j \in (1 \ldots k_q)\}$ for each class $l \in L$ s.t. $Q_l \cap S_l = \{\}$; we define $Q = \bigcup_{l \in L} Q_l$. The support set provides a few labeled examples of new classes not seen at training time that f_ϕ must adapt to i.e. learn to classify, whereas the query set is used for evaluation. The definition of few-shot classification requires that evaluation is done on episodes; however, most few-shot learning methods train as well as evaluate on episodes. Consistent with prior work, we train both MAML and prototypical networks methods on episodes, as opposed to mini-batches.

3.2 Few-shot IC/SF

Few-shot IC/SF extends the prior definition of *few-shot classification* to include both IC and SF tasks.

As Geng et al. (2019) showed, it is straightforward to formulate IC as a *few-shot classification* task. Simply let the class labels y_l in section 3.1 correspond to IC labels and partition the set of ICs into the train and test splits, L_{train} and L_{test}. Building on this few-shot IC formulation, we re-define the *support* and *query* sets to include the slots t_l, in addition the intent y_l, assigned to each example x_l. Thus, the set of *support* and *query* instances for class $l \in L$ become $S_l = \{(x_l^i, t_l^i, y_l)|i \in (1 \ldots k_l)\}$ and $Q_l = \{(x_l^j, t_l^j, y_l)|j \in (1 \ldots k_q)\}$, respectively. To construct an episode, we sample a total of $k_l + k_q$ labeled examples per IC $l \in L$ to form the support and query sets. Since utterances can exhibit many unique slot-label combinations, it is possible to sample an episode such that a slot-label appears in only the query or support set. Therefore, to ensure fair evaluation, we "*mask*" any slot-label that appears in only the query or support set by replacing it with the *Other* slot label, which is ignored by our SF evaluation metric.

4 Approach

4.1 Prototypical Networks for Joint Intent Classification and Slot Filling

The original formulation of prototypical networks (Snell et al., 2017) is not directly applicable to sequence labeling. Accordingly, we extend prototypical networks to perform joint sentence classification and sequence labeling. Our extension computes "prototypes" c_l and c_a for each intent class l and slot-label a, respectively. Each prototype $c \in \mathbb{R}^\mathbb{D}$ is the mean vector of the embeddings belonging to a given intent class or slot-label class. These embeddings are output by a sequence encoder $f_\phi(x) :\to \mathbb{R}^D$, which takes a variable length utterance of m tokens $x^i = (x_1^i, x_2^i, \ldots, x_m^i)$ as input, and outputs the final hidden state $h \in \mathbb{R}^D$ of the encoder. For ease of notation, let $S_l = \{(x_l^i, t_l^i, y_l)\}$ be the support set instances with intent class y_l. And let $S_a = \{(x_{[1:j]}^i, t_{[1:j]}^i, y^i)|t_j^i = a\}$ be the support set sub-sequences with slot-label a for the token x_j^i in x^i. Using this notation, we calculate slot-label and intent class prototypes as follows:

$$c_l = \frac{1}{|S_l|} \sum_{(x^i, t^i, y_l) \in S_l} f_\phi(x^i) \qquad (1)$$

$$c_a = \frac{1}{|S_a|} \sum_{(x_{[1:j]}^i, t_{[1:j]}^i, y^i)} f_\phi(x_{[1:j]}^i) \qquad (2)$$
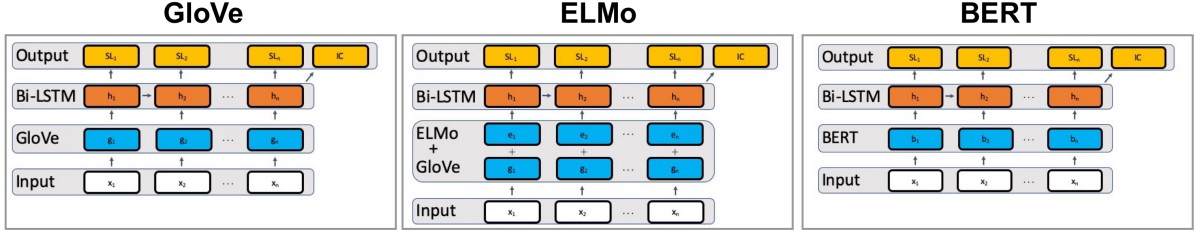
Figure 2: Three model architectures, each consisting of an embedding layer, comprised of either GloVe word embeddings (GloVe), GloVe word embeddings concatenated with ELMo embeddings (ELMo), or BERT embeddings (BERT), that feed into a bi-directional LSTM, which is followed by fully connected intent and slot output layers.

Given an example $(x^*, t^*, y^*) \in Q$, we compute the conditional probability $p(y = l \mid x^*, S)$ that the utterance x^* has intent class l as the normalized Euclidean distance between $f_\phi(x^*)$ and the prototype c_l,

$$p(y = l \mid x^*, S) = \frac{\exp(-\|f_\phi(x^*) - c_l\|_2^2)}{\sum_{l'} \exp(-\|f_\phi(x^*) - c_{l'}\|_2^2)}$$

Similarly, we compute the conditional probability $p(t_j^* = a \mid x^*, S)$ that the j-th token x_j^* in the utterance x^* has slot-label $t_j^* = a$ as the normalized Euclidean distance between $f_\phi(x_{[1:j]}^*)$ and the prototype c_a,

$$p(t_j^* = a \mid x^*, S) = \frac{\exp(-\left\|f_\phi(x_{[1:j]}^*) - c_a\right\|_2^2)}{\sum_{a'} \exp(-\left\|f_\phi(x_{[1:j]}^*) - c_{a'}\right\|_2^2)}$$

We define the joint IC and SF prototypical loss function $\mathcal{L}_{proto}$ as the sum of the IC and SF negative log-likelihoods averaged over the query set instances given the support set:

$$\mathcal{L}_{proto} = \frac{1}{|Q|} \sum_{(x^*, t^*, y^*) \in Q} \mathcal{L}_{protoIC} + \mathcal{L}_{protoSF}$$

$$\mathcal{L}_{protoIC} = -\log p(y = y^* \mid x^*, S)$$

$$\mathcal{L}_{protoSF} = -\sum_{t_j^* \in t^*} \log p(t_j^* = a \mid x^*, S)$$

4.2 Model Agnostic Meta Learning (MAML)

MAML optimizes the parameters ϕ of the encoder f_ϕ such that when ϕ is fine-tuned on the support set S for d steps, $\phi' \leftarrow \text{Finetune}(\phi, d \mid S)$, the fine-tuned model $f_{\phi'}$ generalizes well to new class instances in the query set Q. This is achieved by updating ϕ to minimize the loss of the fine-tuned model $\mathcal{L}(f_{\phi'}, Q)$ on the query set Q. The update

to ϕ takes the form $\phi \leftarrow \phi - \nabla_\phi \mathcal{L}(f_{\phi'}, Q)$, where $\mathcal{L}$ is the sum of IC and SF softmax cross entropy loss functions. Concretely, given a support and query set (S, Q), MAML performs the following two step optimization procedure:

1. $\phi' \leftarrow \text{Finetune}(\phi, d \mid S)$

2. $\phi \leftarrow \phi - \nabla_\phi \mathcal{L}(f_{\phi'}, Q)$

Although, the initial formulation of MAML, which we outline here, utilizes stochastic gradient descent (SGD) to update the initial parameters ϕ, in practice, an alternate gradient based update rule can be used in place of SGD. Empirically, we find it beneficial to use Adam in place of SGD.

A drawback to MAML is that computing the "meta-gradient" $\nabla_\phi \mathcal{L}(f_{\phi'}, Q)$ requires calculating a second derivative, since the gradient must back-propagate through the sequence of updates made by $\text{Finetune}(\phi, d \mid S)$. Fortunately, in the same work where (Finn et al., 2017) introduce MAML, they propose a first order approximation of MAML, foMAML, which ignores these second derivative terms and performs nearly as well as the original method. We utilize foMAML in our experiments to avoid memory issues associated with MAML.

5 Few-shot IC/SF Benchmark

As there is no existing benchmark for *few-shot IC/SF*, we propose few-shot splits for the Air Travel Information System (ATIS, Hemphill et al. (1990)), SNIPS (Coucke et al., 2018), and Task Oriented Parsing (TOP, (Gupta et al., 2018)) datasets. A *few-shot IC/SF* benchmark is beneficial for two reasons. Firstly, the benchmark evaluates generalization across multiple domains. Secondly, researchers can combine these datasets in the future to experiment with larger settings of n-way during training and evaluation.

Split	ATIS				SNIPS				TOP			
	#Utt	#IC	#SL	#SV	#Utt	#IC	#SL	#SV	#Utt	#IC	#SL	#SV
Train	4,373	5	116	461	8,230	4	33	8,549	20,345	7	38	5,574
Dev	662	7	122	260	-	-	-	-	4,333	5	33	2,228
Test	829	7	128	258	6,254	3	20	7,567	4,426	6	39	1,341
Total	5,864	19	366	583	14,484	7	53	13,599	29,104	18	110	6821

Table 1: Statistics on utterance (Utt), intent (IC), slot label (SL), and slot value (SV) counts for ATIS, TOP, and SNIPS few-shot train, development, and test splits as well as the full dataset, provided under the heading total.

5.1 Datasets

ATIS is a well-known dataset for dialog system research, which comprises conversations from the *airline* domain. SNIPS, on the other hand, is a public benchmark dataset developed by the Snips corporation to evaluate the quality of IC and SF services. The SNIPS dataset comprises multiple domains including music, media, and weather. TOP, which pertains to navigation and event search, is unique in that 35% of the utterances contain multiple, nested intent labels. These hierarchical intents require the use of specialized models. Therefore, we utilize only the remaining, non-hierarchical 65% of utterances in TOP. To put the size and diversity of these datasets in context, we provide utterance, intent, slot-label, and slot value counts for each dataset in table 1.

5.2 Few-shot Splits

We target train, development, and test split sizes of 70%, 15%, and 15%, respectively. However, the ICs in these datasets are highly imbalanced, which prevents us from hitting these targets exactly. Thereby, we manually select the ICs to include in each split. For the SNIPS dataset, we choose not to form a development split because there are only 7 ICs in the SNIPS dataset, and we require a minimum of 3 ICs per split. During preprocessing we modify slot label names by adding the associated IC as a prefix to each slot. This preprocessing step ensures that the slot labels are no longer pure named entities, but specific semantic roles in the context of particular intents. In table 1, we provide statistics on the few-shot splits for each dataset.

6 Experiments

6.1 Episode Construction

For train and test episodes, we sample both the the number of classes in each episode, the "way" n, and the number of examples to include for each sampled class l, the class "shot" k_l, using the procedure put forward in (Triantafillou et al., 2019). By sampling the shot and way, we allow for unbalanced support sets and a variable number of classes per episode. These allowances are compatible with the large degree of class imbalances present in our benchmark, which would make it difficult to apply a fixed shot and way for all intents.

To construct an episode given a few-shot class split L_{split}, we first sample the way n uniformly from the range $[3, |L_{split}|]$. We then sample n intent classes uniformly at random from L_{split} to form L. Next, we sample the query shot k_q for the episodes as follows:

$$k_q = \min(10, (\min_{l \in L} [0.5 * |X_l|]))$$

where X_l is the set of examples with class label l. Given the query shot k_q, we compute the target support set size for the episode as:

$$|S| = \min(K_{\max}, \sum_{l \in L} \lceil \beta \min(20, |X_l| - q) \rceil)$$

where β is sampled uniformly from the range $(0, 1]$ and $K_{\max}$ is the maximum episode size. Lastly, we sample the support shot k_l for each class as:

$$k_l = \min(\lfloor R_l * (|S| - |L|) \rfloor + 1, |X_l| - q)$$

where R_l is a noisy estimate of the normalized proportion of the dataset made up by class l, which we compute as follows:

$$R_l = \frac{\exp(\alpha_l) * |X_l|}{\sum_{l' \in L} \exp(\alpha_{l'}) * |X_{l'}|)}$$

The noise in our estimate of the proportion R_l is introduced by sampling the value of α_l uniformly from the interval $[\log(0.5), \log(2))$.

6.2 Episode Sizes

We present IC/SF results for two settings of maximum episode size, $K_{\max} = 20$ and $K_{\max} = 100$, in tables 2/4 and 3/5, respectively. When the maximum episode size $K_{\max} = 20$, the average support set shot k_l is 3.58 for ATIS, 3.78 for TOP, and 5.22 for SNIPS. In contrast, setting the maximum episode size to $K_{\max} = 100$ increases the average support set shot k_l to 9.15 for ATIS, 9.81 for TOP, and 10.83 for SNIPS.

6.3 Training Settings

In our experiments, we consider two training settings. One in which we train on episodes, or batches in the case of our baseline, from a single dataset. And another, *joint* training approach that randomly selects the dataset from which to sample a given episode/batch. After sampling an episode, we remove its contents from a buffer of available examples. If there are no longer enough examples in the buffer to create an episode, we refresh the buffer to contain all examples.

6.4 Network Architecture

We evaluate the network architectures depicted in Figure 2. These networks consist of an embedding layer, a sequence encoder, and two output layers for slots and intents, respectively. We greedily predict the slot-label for each token in the input sequence, according to the maximum output logit at each position. We plan to explore alternate search algorithms (e.g., beam search) in future work.

Each architecture uses a different pre-trained embedding layer type, which are either non-contextual or contextual. We experiment with one non-contextual embedding, GLOVE word vectors (Pennington et al., 2014), as well as two contextual embeddings, GLOVE concatenated with ELMO embeddings (Peters et al., 2018), and BERT embeddings (Devlin et al., 2018). The sequence encoder is a bi-directional LSTM (Hochreiter and Schmidhuber, 1997) with a 512-dimensional hidden state. Output layers are fully connected and take concatenated forward and backward LSTM hidden states as input. Pre-trained embeddings are kept frozen for training and adaptation. Attempts to fine-tune BERT led to inferior results. We refer to each architecture by its embedding type, namely GLOVE, ELMO, or BERT.

6.5 Baseline

We compare the performance of our approach against a FINE-TUNE baseline, which implements the domain adaptation framework commonly applied to low resource IC/SF (Goyal et al., 2018). We pre-train the FINE-TUNE baseline, either jointly or individually, on the classes in our training split(s). Then at evaluation time, we freeze the pre-trained encoder and "fine-tune" new output layers for the slots and intents included in the support set. This fine-tuned model is then used to predict the intent and slots for each held out example in the query set.

6.6 Hyper-parameters

We train all models using the Adam optimizer (Kingma and Ba, 2014). We use the default learning rate of 0.001 for the baseline and prototypical networks. For foMAML we set the outer learning rate to 0.0029 and finetune for $d = 8$ steps with an inner learning rate of 0.01. We pre-train the FINE-TUNE baseline with a batch size of 512. At test time, we fine-tune the baseline for 10 steps on the support set. We train the models without contextual embeddings (GloVe alone) for 50 epochs and those with contextual ELMo or BERT embeddings for 30 epochs because they exhibit faster convergence.

6.7 Evaluation Metrics

To assess the performance of our models, we report the average IC accuracy and slot F1 score over 100 episodes sampled from the test split of an individual dataset. We use the AllenNLP (Gardner et al., 2017) CategoricalAccuracy implementation to compute IC Accuracy. And to compute slot F1 score, we use the seqeval library's span based F1 score implementation.[2] The span based F1 score is a relatively harsh metric in the sense that a slot label prediction is only considered correct if the slot label and span exactly match the ground truth annotation.

7 Results

7.1 Few-shot Learning Algorithms

Prototypical networks Considering both IC and SF tasks, prototypical networks is the best performing algorithm. The most successful variant of prototypical networks, Proto ELMO + *joint* training, obtains absolute improvements over the FINE-TUNE

---

[2]https://github.com/chakki-works/seqeval

Embed.	Algorithm	IC Accuracy					
		SNIPS	SNIPS (joint)	ATIS	ATIS (joint)	TOP	TOP (joint)
GloVe	Fine-tune	69.52 ± 2.88	70.25 ± 1.85	49.50 ± 0.65	58.26 ± 1.12	37.58 ± 0.54	40.93 ± 2.77
GloVe	foMAML	61.08 ± 1.50	59.67 ± 2.12	54.66 ± 1.82	45.20 ± 1.47	33.75 ± 1.30	31.48 ± 0.50
GloVe	Proto	68.19 ± 1.76	68.77 ± 1.60	65.46 ± 0.81	63.91 ± 1.27	43.20 ± 0.85	38.65 ± 1.35
ELMo	Fine-tune	**85.53 ± 0.35**	**87.64 ± 0.73**	49.25 ± 0.74	58.69 ± 1.56	45.49 ± 0.61	47.63 ± 2.75
ELMo	foMAML	78.90 ± 0.77	78.86 ± 1.31	53.90 ± 0.96	52.47 ± 2.86	38.67 ± 1.02	36.49 ± 0.99
ELMo	Proto	83.54 ± 0.40	85.75 ± 1.57	**65.95 ± 2.29**	**65.19 ± 1.29**	50.57 ± 2.81	50.64 ± 2.72
BERT	Fine-tune	76.04 ± 8.84	77.53 ± 5.69	43.76 ± 4.61	50.73 ± 3.86	39.21 ± 3.09	40.86 ± 3.75
BERT	foMAML	67.36 ± 1.03	68.37 ± 0.48	50.27 ± 0.69	48.80 ± 2.82	38.50 ± 0.43	36.20 ± 1.21
BERT	Proto	81.39 ± 1.85	81.44 ± 2.91	58.84 ± 1.33	58.82 ± 1.55	**52.76 ± 2.26**	**52.64 ± 2.58**

Table 2: $K_{\max} = 20$ average IC accuracy on 100 test episodes from the ATIS, SNIPS, or TOP datasets in the form mean $\pm$ standard deviation, computed over 3 random seeds, comparing GloVe, ELMo, and BERT model variants for both individual and *joint* training, where we train on all training sets and test on a specific test set.

Embed.	Algorithm	IC Accuracy					
		SNIPS	SNIPS (joint)	ATIS	ATIS (joint)	TOP	TOP (joint)
GloVe	Fine-tune	72.24 ± 2.58	73.00 ± 1.84	49.91 ± 1.90	56.07 ± 2.94	39.66 ± 1.34	41.10 ± 0.65
GloVe	foMAML	66.75 ± 1.28	67.34 ± 2.62	54.92 ± 0.87	58.46 ± 1.91	33.62 ± 1.53	35.68 ± 0.62
GloVe	Proto	70.45 ± 0.49	72.66 ± 1.96	**70.25 ± 0.39**	69.58 ± 0.41	48.84 ± 1.59	46.85 ± 0.86
ELMo	Fine-tune	**87.69 ± 1.05**	**88.90 ± 0.18**	49.42 ± 0.79	56.99 ± 2.12	47.44 ± 1.61	48.87 ± 0.54
ELMo	foMAML	80.80 ± 0.47	81.62 ± 1.07	59.10 ± 2.52	56.16 ± 1.34	41.80 ± 1.49	36.24 ± 0.79
ELMo	Proto	86.76 ± 1.62	87.74 ± 1.08	70.10 ± 1.26	**71.89 ± 1.45**	58.60 ± 1.91	56.87 ± 0.39
BERT	Fine-tune	76.66 ± 8.68	79.53 ± 4.25	44.08 ± 6.05	49.71 ± 3.84	40.05 ± 2.35	40.46 ± 1.74
BERT	foMAML	70.43 ± 1.56	72.79 ± 1.11	51.36 ± 3.74	50.25 ± 0.88	36.15 ± 2.17	35.24 ± 0.35
BERT	Proto	83.51 ± 0.88	86.29 ± 1.09	66.89 ± 2.31	65.70 ± 2.31	**61.30 ± 0.32**	**62.51 ± 1.79**

Table 3: $K_{\max} = 100$ average IC accuracy on 100 test episodes from the ATIS, SNIPS, or TOP datasets in the form mean $\pm$ standard deviation, computed over 3 random seeds, comparing GloVe, ELMo, and BERT model variants for both individual and *joint* training, where we train on all training sets and test on a specific test set.

Embed.	Algorithm	Slot F1 Measure					
		SNIPS	SNIPS (joint)	ATIS	ATIS (joint)	TOP	TOP (joint)
GloVe	Fine-tune	6.72 ± 1.24	6.68 ± 0.40	2.57 ± 1.21	13.22 ± 1.07	0.90 ± 0.51	0.76 ± 0.21
GloVe	foMAML	14.07 ± 1.01	12.91 ± 0.43	18.44 ± 0.91	16.91 ± 0.32	5.34 ± 0.43	9.22 ± 1.03
GloVe	Proto	29.63 ± 0.75	27.75 ± 2.52	31.19 ± 1.15	38.45 ± 0.97	10.65 ± 0.83	18.55 ± 0.35
ELMo	Fine-tune	22.02 ± 1.13	16.00 ± 2.07	7.47 ± 2.60	7.19 ± 1.71	1.26 ± 0.46	1.17 ± 0.32
ELMo	foMAML	33.81 ± 0.33	32.82 ± 0.84	27.58 ± 1.25	24.45 ± 1.20	**22.35 ± 1.23**	15.53 ± 0.64
ELMo	Proto	**59.88 ± 0.53**	**59.73 ± 1.72**	33.97 ± 0.38	**40.90 ± 2.21**	20.12 ± 0.25	**28.97 ± 0.82**
BERT	Fine-tune	12.47 ± 0.31	8.75 ± 0.28	9.24 ± 1.67	15.93 ± 3.10	3.15 ± 0.28	1.08 ± 0.30
BERT	foMAML	12.72 ± 0.12	13.28 ± 0.53	18.91 ± 1.01	16.05 ± 0.32	5.93 ± 0.43	8.23 ± 0.81
BERT	Proto	42.09 ± 1.11	43.77 ± 0.54	**37.61 ± 0.82**	39.27 ± 1.84	20.81 ± 0.40	28.24 ± 0.53

Table 4: $K_{\max} = 20$ average Slot F1 score on 100 test episodes from the ATIS, SNIPS, or TOP datasets in the form mean $\pm$ standard deviation, computed over 3 random seeds, comparing GloVe, ELMo, and BERT model variants for both individual and *joint* training, where we train on all training sets and test on a specific test set.

Embed.	Algorithm	Slot F1 Measure					
		SNIPS	SNIPS (joint)	ATIS	ATIS (joint)	TOP	TOP (joint)
GloVe	Fine-tune	7.06 ± 1.87	7.76 ± 0.91	2.72 ± 1.65	17.20 ± 3.03	1.26 ± 0.44	0.67 ± 0.33
GloVe	foMAML	16.77 ± 0.67	16.53 ± 0.32	17.80 ± 0.42	23.33 ± 2.89	4.11 ± 0.81	9.89 ± 1.13
GloVe	Proto	31.57 ± 1.28	31.17 ± 1.31	31.32 ± 2.79	41.07 ± 1.14	9.99 ± 1.08	18.93 ± 0.77
ELMo	Fine-tune	22.37 ± 0.91	17.09 ± 2.57	8.93 ± 2.86	11.09 ± 2.00	2.04 ± 0.41	1.03 ± 0.24
ELMo	foMAML	36.10 ± 1.49	37.33 ± 0.24	26.91 ± 2.64	26.37 ± 0.15	18.32 ± 0.52	16.55 ± 0.79
ELMo	Proto	**62.71 ± 0.40**	**62.14 ± 0.75**	35.20 ± 2.46	41.28 ± 2.73	18.44 ± 2.41	28.33 ± 1.33
BERT	Fine-tune	14.71 ± 0.43	10.50 ± 0.90	11.53 ± 1.46	20.41 ± 1.85	4.98 ± 0.66	1.48 ± 0.85
BERT	foMAML	14.99 ± 1.29	15.83 ± 0.94	17.68 ± 2.42	17.11 ± 1.31	3.37 ± 0.36	10.58 ± 0.45
BERT	Proto	46.50 ± 0.75	48.77 ± 0.71	**40.63 ± 3.37**	**43.10 ± 1.76**	**20.58 ± 2.27**	**28.92 ± 1.09**

Table 5: $K_{\max} = 100$ average Slot F1 score on 100 test episodes from the ATIS, SNIPS, or TOP datasets in the form mean $\pm$ standard deviation, computed over 3 random seeds, comparing GloVe, ELMo, and BERT model variants for both individual and *joint* training, where we train on all training sets and test on a specific test set.

ELMO + *joint* training baseline of up to 6% IC accuracy and 43 slot F1 points for $K_{max} = 20$, and 14% IC accuracy and 45 slot F1 points for $K_{max} = 100$. The one case in which Proto ELMO + *joint* training does worse than the baseline is on SNIPS IC, but these losses are all under 2%. Over the 36 possible combinations of dataset, embedding type, and setting of K_{max}, Prototypical networks performs best in 27/36 instances for IC and 35/36 instances for SL. In comparison, FINE-TUNE performs best in 9/36 instances for IC and is never the best algorithm for SL. Conversely, foMAML is never the best algorithm for IC and is the best algorithm in 1/36 cases for SL.

foMAML The results for foMAML are more mixed in terms of IC and SF performance relative to the baseline. The best foMAML variant, foMAML ELMO, underperforms FINE-TUNE ELMO on SNIPS and TOP IC by up to 6%. Yet foMAML improves IC accuracy by 4% ($K_{max} = 20$) to 9% ($K_{max} = 100$) on ATIS. foMAML ELMO consistently outperforms FINE-TUNE ELMO on SF for all datasets, generating gains of 11~21 F1 points for $K_{max} = 20$ and 13~17 F1 points for $K_{max} = 100$. Notably, BERT and foMAML do not work well in combination. Specifically, the SF performance of foMAML BERT is comparable to, or worse than, foMAML GLOVE on all datasets for both $K_{max} = 20$ and $K_{max} = 100$.

7.2 Model Variants

Non-contextual Pretrained Embeddings The GLOVE model architecture does not perform as well as ELMO or BERT. On average over experimental settings, the GLOVE variant of the winning algorithm has 10% lower IC Accuracy and 16 point lower slot F1 score than the winning algorithm paired with the best model. Note that an experimental setting here refers to a combination of dataset, value of K_{max}, and use of individual or joint training. Somewhat surprisingly, GLOVE performs nearly as well as ELMO and even better than BERT on ATIS IC. We speculate that ATIS IC does not benefit as much from the use of ELMO or BERT because ATIS carrier phrases are less diverse, as evidenced by the smaller number of unique carrier phrases in the ATIS test set (527) compared to SNIPS (3,718) and TOP (4,153).

Contextual Pretrained Embeddings A priori, it is reasonable to suspect that the performance

gain obtained by our few-shot learning algorithms could be dwarfed by the benefit of using a large, pre-trained model like ELMO or BERT. However, our experimental results suggest that the use of pre-trained language models is complementary to our approach, in most cases. For example, ELMO increases the slot F1 score of foMAML from 14.07 to 33.81 and boosts the slot F1 of prototypical networks from 31.57 to 62.71 on the SNIPS dataset for $K_{max} = 100$. Similarly, when $K_{max} = 20$, BERT improves foMAML and prototypical networks TOP IC accuracy from 33.75% to 38.50% and from 43.20% to 52.76%, respectively. In aggregate, we find ELMO outperforms BERT. We quantify this via the average absolute improvement ELMO obtains over BERT when both models use the winning algorithm for a given dataset and training setting. On average, ELMO improves IC accuracy over BERT by 2% for $K_{max} = 20$ and 1% for $K_{max} = 100$. With respect to slot F1 score, ELMO produces an average gain over BERT of 5 F1 points for $K_{max} = 20$ and 3 F1 points for $K_{max} = 100$. This is consistent with previous findings in (Peters et al., 2019) that ELMO can outperform BERT on certain tasks when the models are kept frozen and not fine-tuned.

7.3 Joint Training

Few-shot learning algorithms are in essence learning to learn new classes. Therefore, these algorithms should be adept at leveraging a diverse training dataset to improve generalization. We test this hypothesis by jointly training each approach on all three datasets. Our results demonstrate that joint training has little effect on IC Accuracy; however, it improves the SF performance of prototypical networks, particularly on ATIS and TOP. Joint training increases Prototypical networks average slot F1 score, computed over datasets and model variants, by 4.41 points from 31.77 to 36.18 for $K_{max} = 20$ and by 5.20 points from 32.99 to 38.19 when $K_{max} = 100$. In comparison, Fine-tune obtains much smaller average absolute improvements, 0.55 F1 points and 1.29 F1 points for $K_{max} = 20$ and $K_{max} = 100$, respectively.

8 Conclusion

We show that few-shot learning techniques can substantially improve IC/SF performance in ultra low resource scenarios. Specifically, our extension of prototypical networks for joint IC and SF consis-

tently outperforms a fine-tuning baseline with respect to both IC Accuracy and slot F1 score. Moreover, we establish a benchmark for *few-shot IC/SF* to support future work on this important topic. Our contribution is a step toward the creation of more sample efficient IC/SF models. Yet there is still considerable work to be done in pursuit of this goal. In particular, we encourage the creation of larger *few-shot IC/SF* benchmarks to test how few-shot learning algorithms scale with larger episode sizes.

References

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Maël Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *CoRR*, abs/1805.10190.

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

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. 2019. Investigating meta-learning algorithms for low-resource natural language understanding tasks. *arXiv preprint arXiv:1908.10423*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Alexander Fritzler, Varvara Logacheva, and Maksim Kretov. 2019. Few-shot classification in named entity recognition task. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 993–1000. ACM.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Ruiying Geng, Binhua Li, Yongbin Li, Yuxiao Ye, Ping Jian, and Jian Sun. 2019. Few-shot text classification with induction network. *arXiv preprint arXiv:1902.10482*.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757.

Anuj Goyal, Angeliki Metallinou, and Spyros Matsoukas. 2018. Fast and scalable expansion of natural language understanding functionality for intelligent agents. *arXiv preprint arXiv:1805.01542*.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Arshit Gupta, John Hewitt, and Katrin Kirchhoff. 2019. Simple, fast, accurate intent classification and slot labeling. *arXiv preprint arXiv:1903.08268*.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

Dilek Hakkani-Tür, Gökhan Tür, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Interspeech*, pages 715–719.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yutai Hou, Zhihan Zhou, Yijia Liu, Ning Wang, Wanxiang Che, Han Liu, and Ting Liu. 2019. Few-shot sequence labeling with label dependency transfer. *arXiv preprint arXiv:1906.08711*.

Xiang Jiang, Mohammad Havaei, Gabriel Chartrand, Hassan Chouaib, Thomas Vincent, Andrew Jesson, Nicolas Chapados, and Stan Matwin. 2018. Attentive task-agnostic meta-learning for few-shot text classification.

Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. 2017. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Gregory Koch. 2015. Siamese neural networks for one-shot image recognition.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2017. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy. Association for Computational Linguistics.

Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850.

Sebastian Schuster, Sonal Gupta, Rushin Shah, and Mike Lewis. 2018. Cross-lingual transfer learning for multilingual task oriented dialog. *arXiv preprint arXiv:1810.13327*.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087.

Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. 2019. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018. Diverse few-shot text classification with multiple metrics. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1206–1215.

A Appendices

A.1 Few-shot IC/SF Splits

We list the few-shot splits that we establish for the ATIS, SNIPS, and TOP datasets in tables 6, 7, 8, respectively. In addition to the assignment of intent classes to train, development (dev) and test splits, we also report the number of utterances and slot labels associated with each intent class.

Split	IC Name	# Utt	# SL	SL Names (without IC prefix)
Train	atis_flight	4298	71	or, mod, meal, flight, economy, connect, day_name, city_name, round_trip, class_type, flight_mod, compartment, flight_stop, flight_days, flight_time, fare_amount, airline_name, airline_code, airport_name, airport_code, cost_relative, aircraft_code, flight_number, period_of_day, toloc.city_name, arrive_time.time, meal_description, toloc.state_name, depart_time.time, toloc.state_code, depart_date.year, fare_basis_code, fromloc.city_name, stoploc.city_name, fromloc.state_name, toloc.airport_name, fromloc.state_code, toloc.country_name, toloc.airport_code, stoploc.state_code, fromloc.airport_name, fromloc.airport_code, stoploc.airport_name, stoploc.airport_code, depart_date.day_name, arrive_date.day_name, depart_time.end_time, arrive_time.end_time, return_date.day_name, arrive_date.month_name, arrive_date.day_number, depart_date.month_name, depart_time.start_time, arrive_time.start_time, depart_time.period_mod, arrive_time.period_mod, return_time.period_mod, return_date.month_name, return_date.day_number, arrive_time.time_relative, depart_time.time_relative, depart_date.day_number, return_date.date_relative, arrive_date.date_relative, depart_time.period_of_day, depart_date.today_relative, arrive_date.today_relative, arrive_time.period_of_day, return_time.period_of_day, return_date.today_relative
Train	atis_capacity	37	5	mod, airline_name, aircraft_code, toloc.city_name, fromloc.city_name
Train	atis_flight_no	20	22	or, flight_mod, class_type, flight_time, airline_name, cost_relative, flight_number, toloc.city_name, arrive_time.time, toloc.state_name, depart_time.time, toloc.state_code, fromloc.city_name, stoploc.city_name, fromloc.state_name, depart_date.day_name, depart_date.month_name, depart_date.day_number, arrive_time.time_relative, depart_time.time_relative, depart_time.period_of_day, depart_date.today_relative
Train	atis_meal	12	12	meal, airline_name, airline_code, flight_number, toloc.city_name, arrive_time.time, toloc.state_code, meal_description, fromloc.city_name, toloc.airport_code, depart_date.day_name, depart_time.period_of_day
Train	atis_restriction	6	6	round_trip, fare_amount, cost_relative, toloc.city_name, restriction_code, fromloc.city_name
Dev	atis_airfare	471	45	or, meal, economy, connect, round_trip, class_type, flight_mod, fare_amount, flight_stop, flight_time, flight_days, airline_name, airline_code, cost_relative, flight_number, aircraft_code, toloc.city_name, depart_time.time, toloc.state_code, toloc.state_name, depart_date.year, arrive_time.time, fromloc.city_name, stoploc.city_name, toloc.airport_name, fromloc.state_name, toloc.airport_code, fromloc.state_code, fromloc.airport_name, fromloc.airport_code, depart_date.day_name, arrive_date.day_name, arrive_date.month_name, arrive_date.day_number, depart_date.month_name, depart_date.day_number, return_date.month_name, return_date.day_number, depart_time.period_mod, depart_time.time_relative, depart_date.date_relative, arrive_time.time_relative, arrive_date.date_relative, depart_time.period_of_day, depart_date.today_relative
Dev	atis_flight_time	55	20	flight_mod, class_type, flight_time, airline_name, airline_code, airport_name, flight_number, aircraft_code, toloc.city_name, depart_time.time, meal_description, fromloc.city_name, toloc.airport_code, fromloc.airport_name, depart_date.day_name, depart_date.month_name, depart_date.day_number, depart_date.date_relative, depart_time.time_relative, depart_time.period_of_day
Dev	atis_quantity	54	25	economy, city_name, class_type, round_trip, flight_stop, flight_days, airline_code, airline_name, flight_number, aircraft_code, toloc.city_name, arrive_time.time, fare_basis_code, depart_time.time, fromloc.city_name, stoploc.city_name, toloc.airport_name, arrive_date.month_name, arrive_date.day_number, depart_date.month_name, depart_date.day_number, arrive_time.time_relative, depart_time.time_relative, depart_date.today_relative, depart_time.period_of_day
Dev	atis_distance	30	8	city_name, airport_name, toloc.city_name, depart_time.time, fromloc.city_name, fromloc.airport_name, depart_date.month_name, depart_date.day_number
Dev	atis_city	25	11	city_name, class_type, airline_name, airport_code, airport_name, toloc.city_name, depart_time.time, fromloc.city_name, fromloc.airport_code, depart_time.time_relative, depart_time.period_of_day
Dev	atis_ground_fare	25	6	city_name, airport_name, transport_type, toloc.city_name, fromloc.city_name, fromloc.airport_name
Dev	atis_airline;atis_flight_no	2	7	toloc.city_name, arrive_time.time, fromloc.city_name, depart_date.month_name, depart_date.day_number, depart_date.date_relative, arrive_time.time_relative
Test	atis_ground_service	291	23	or, time, day_name, city_name, state_code, state_name, month_name, day_number, flight_time, airport_name, airport_code, time_relative, transport_type, today_relative, period_of_day, toloc.city_name, fromloc.city_name, toloc.airport_name, fromloc.airport_name, depart_date.day_name, depart_date.month_name, depart_date.day_number, depart_date.date_relative
Test	atis_airline	195	36	mod, connect, city_name, class_type, round_trip, flight_stop, flight_days, airline_code, airport_name, airline_name, aircraft_code, flight_number, cost_relative, toloc.city_name, toloc.state_code, depart_time.time, arrive_time.time, toloc.state_name, fromloc.city_name, stoploc.city_name, toloc.airport_name, fromloc.state_code, fromloc.airport_code, fromloc.airport_name, depart_date.day_name, depart_time.end_time, arrive_date.month_name, arrive_date.day_number, depart_date.month_name, depart_date.day_number, depart_time.start_time, depart_time.time_relative, depart_date.date_relative, depart_date.today_relative, depart_time.period_of_day, arrive_time.period_of_day
Test	atis_abbreviation	180	14	mod, meal, meal_code, days_code, class_type, airport_code, airline_code, airline_name, aircraft_code, booking_class, toloc.city_name, fare_basis_code, restriction_code, fromloc.city_name
Test	atis_aircraft	90	23	mod, city_name, class_type, flight_mod, airline_name, airline_code, flight_number, aircraft_code, toloc.city_name, depart_time.time, toloc.state_code, arrive_time.time, fromloc.city_name, stoploc.city_name, depart_date.day_name, arrive_date.day_name, depart_date.month_name, depart_date.day_number, arrive_date.month_name, arrive_date.day_number, depart_time.time_relative, arrive_time.time_relative, depart_time.period_of_day
Test	atis_airport	38	9	mod, city_name, state_code, state_name, flight_stop, airport_name, airline_name, toloc.city_name, fromloc.city_name
Test	atis_flight;atis_airfare	33	21	flight_mod, round_trip, fare_amount, flight_stop, airline_name, flight_number, cost_relative, toloc.city_name, depart_time.time, arrive_time.time, toloc.state_code, fromloc.city_name, depart_date.day_name, return_date.day_name, depart_date.month_name, depart_date.day_number, depart_date.date_relative, arrive_time.time_relative, return_date.date_relative, depart_time.period_of_day
Test	atis_day_name	2	2	toloc.city_name, fromloc.city_name

Table 6: Few-shot splits for the ATIS dataset, listing the split assignment, number of utterances (# Utt), number of slot labels (# SL), and names of slot labels associated with each intent class (IC). For brevity, we exclude the intent class prefix that we add to the slot label names during prepossessing (in the form intent-name/slot-name).

Split	IC Name	# Utt	# SL	SL Names (without IC prefix)
Train	BookRestaurant	2073	14	poi, sort, city, state, country, cuisine, facility, timeRange, served_dish, restaurant_type, restaurant_name, spatial_relation, party_size_number, party_size_description
Train	SearchScreeningEvent	2059	7	timeRange, movie_name, movie_type, object_type, location_name, spatial_relation, object_location_type
Train	RateBook	2056	7	best_rating, rating_unit, object_name, object_type, rating_value, object_select, object_part_of_series_type
Train	AddToPlaylist	2042	5	artist, playlist, music_item, entity_name, playlist_owner
Test	PlayMusic	2100	9	sort, year, album, genre, track, artist, service, playlist, music_item
Test	GetWeather	2100	9	city, state, country, timeRange, geographic_poi, spatial_relation, current_location, condition_temperature, condition_description
Test	SearchCreativeWork	2054	2	object_name, object_type

Table 7: Few-shot splits for the SNIPS dataset, listing the split assignment, number of utterances (# Utt), number of slot labels (# SL), and names of slot labels associated with each intent class (IC). For brevity, we exclude the intent class prefix that we add to the slot label names during prepossessing (in the form intent-name/slot-name).

Split	IC Name	# Utt	# SL	SL Names (without IC prefix)
Train	IN:GET_EVENT	10063	9	SL:AMOUNT, SL:ORDINAL, SL:LOCATION, SL:DATE_TIME, SL:NAME_EVENT, SL:CATEGORY_EVENT, SL:ATTENDEE_EVENT, SL:ATTRIBUTE_EVENT, SL:ORGANIZER_EVENT
Train	IN:GET_INFO_TRAFFIC	8629	11	SL:PATH, SL:SOURCE, SL:LOCATION, SL:WAYPOINT, SL:DATE_TIME, SL:DESTINATION, SL:PATH_AVOID, SL:METHOD_TRAVEL, SL:ROAD_CONDITION, SL:WAYPOINT_AVOID, SL:OBSTRUCTION_AVOID
Train	IN:UNSUPPORTED	1484	0	
Train	IN:GET_ESTIMATED_DEPARTURE	160	10	SL:PATH, SL:SOURCE, SL:WAYPOINT, SL:LOCATION, SL:DESTINATION, SL:METHOD_TRAVEL, SL:ROAD_CONDITION, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE
Train	IN:UNINTELLIGIBLE	4	0	
Train	IN:GET_EVENT_ATTENDEE_AMOUNT	3	5	SL:ORDINAL, SL:LOCATION, SL:DATE_TIME, SL:NAME_EVENT, SL:CATEGORY_EVENT
Train	IN:GET_EVENT_ORGANIZER	2	3	SL:LOCATION, SL:DATE_TIME, SL:CATEGORY_EVENT
Dev	IN:GET_ESTIMATED_DURATION	2309	12	SL:PATH, SL:SOURCE, SL:WAYPOINT, SL:DATE_TIME, SL:DESTINATION, SL:PATH_AVOID, SL:METHOD_TRAVEL, SL:WAYPOINT_AVOID, SL:ROAD_CONDITION, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE
Dev	IN:GET_DISTANCE	1962	9	SL:PATH, SL:SOURCE, SL:AMOUNT, SL:DESTINATION, SL:PATH_AVOID, SL:UNIT_DISTANCE, SL:METHOD_TRAVEL, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_DEPARTURE
Dev	IN:GET_LOCATION	47	5	SL:LOCATION, SL:POINT_ON_MAP, SL:LOCATION_USER, SL:LOCATION_MODIFIER, SL:CATEGORY_LOCATION
Dev	IN:GET_INFO_ROUTE	13	5	SL:PATH, SL:SOURCE, SL:WAYPOINT, SL:DESTINATION, SL:DATE_TIME_DEPARTURE
Dev	IN:GET_EVENT_ATTENDEE	2	2	SL:ATTENDEE_EVENT, SL:CATEGORY_EVENT
Test	IN:UNSUPPORTED_NAVIGATION	2175	0	
Test	IN:GET_DIRECTIONS	752	12	SL:PATH, SL:SOURCE, SL:WAYPOINT, SL:DESTINATION, SL:PATH_AVOID, SL:METHOD_TRAVEL, SL:WAYPOINT_AVOID, SL:ROAD_CONDITION, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE, SL:ROAD_CONDITION_AVOID
Test	IN:GET_ESTIMATED_ARRIVAL	538	11	SL:PATH, SL:SOURCE, SL:WAYPOINT, SL:LOCATION, SL:DESTINATION, SL:PATH_AVOID, SL:METHOD_TRAVEL, SL:ROAD_CONDITION, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE
Test	IN:UNSUPPORTED_EVENT	424	0	
Test	IN:GET_INFO_ROAD_CONDITION	316	9	SL:PATH, SL:SOURCE, SL:LOCATION, SL:DATE_TIME, SL:DESTINATION, SL:METHOD_TRAVEL, SL:ROAD_CONDITION, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE
Test	IN:UPDATE_DIRECTIONS	221	7	SL:PATH, SL:SOURCE, SL:DESTINATION, SL:PATH_AVOID, SL:OBSTRUCTION_AVOID, SL:DATE_TIME_ARRIVAL, SL:DATE_TIME_DEPARTURE

Table 8: Few-shot splits for the TOP dataset, listing the split assignment, number of utterances (# Utt), number of slot labels (# SL), and names of slot labels associated with each intent class (IC). For brevity, we exclude the intent class prefix that we add to the slot label names during prepossessing (in the form intent-name/slot-name).