LN-Annote: An Alternative Approach to Information Extraction from Emails using Locally-Customized Named-Entity Recognition

YoungHoon Jung
Columbia University
jung@cs.columbia.edu

Karl Stratos
Columbia University
stratos@cs.columbia.edu

Luca P. Carloni
Columbia University
luca@cs.columbia.edu

ABSTRACT

Personal mobile devices offer a growing variety of personalized services that enrich considerably the user experience. This is made possible by increased access to personal information, which to a large extent is extracted from user email messages and archives. There are, however, two main issues. First, currently these services can be offered only by large web-service companies that can also deploy email services. Second, keeping a large amount of structured personal information on the cloud raises privacy concerns. To address these problems, we propose LN-Annote, a new method to extract personal information from the email that is locally available on mobile devices (without remote access to the cloud). LN-Annote enables third-party service providers to build a question-answering system on top of the local personal information without having to own the user data. In addition, LN-Annote mitigates the privacy concerns by keeping the structured personal information directly on the personal device. Our method is based on a named-entity recognizer trained in two separate steps: first using a common dataset on the cloud and then using a personal dataset in the mobile device at hand. Our contributions include also the optimization of the implementation of LN-Annote: in particular, we implemented an OpenCL version of the custom-training algorithm to leverage the Graphic Processing Unit (GPU) available on the mobile device. We present an extensive set of experiment results: beside proving the feasibility of our approach, they demonstrate its efficiency in terms of the named-entity extraction performance as well as the execution speed and the energy consumption spent in mobile devices.

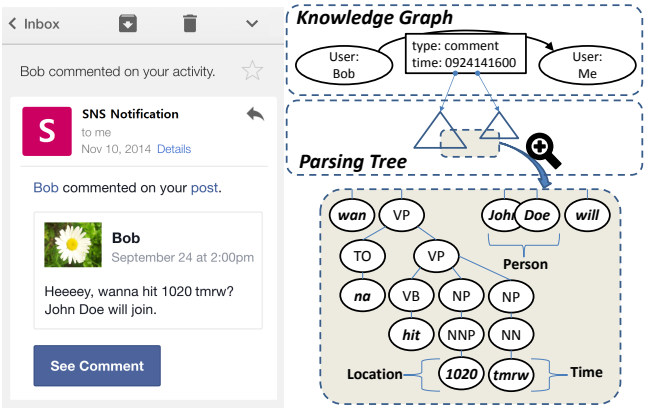
Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services.

Keywords

NER; SNS; Personal Search; Information Extraction; Neural Network; OpenCL.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741633.



(a) An SNS Notification Email (b) Knowledge Graph and Parse Tree

Fig. 1: Parsing email to collect personal information.

1. INTRODUCTION

Recent advancements in personalized web-based services have enriched our daily lives. Intelligent personal assistant services such as Google Now [6] or Apple's Siri [7] can give "directions to home" or alert that "it's time to leave for your next meeting". Meanwhile, personal search services can answer queries based on the user's personal information. Googling "my flights", for instance, produces the upcoming flight reservations that the user has made. Personalized advertisement is another important (and most profitable) instance of personalized web services. The advertisement systems of Amazon, Facebook and Google [1, 4, 5] are known to utilize viewers' personal information such as previous purchase history.

What makes all these personalized services possible? The personal information collected by the service providers. Since its quality determines the quality of the personalized services, web service providers put in significant efforts to improve and extend its collection. One vast source of personal information is found in users' emails. Large web-service companies that provide also email services (like Google, Microsoft, and Yahoo) have the means to offer rich personalized services precisely thanks to the personal information they extract from the users' emails. The example of Fig. 1 illustrates this process. A notification email of a message posted on a Social Network Service (SNS) account by one of the user's friend is parsed through a sequence of steps to build structured data, including: 1) a knowledge graph indicating the subject, the object, a type of the action, and the contents of the comment; 2) the parsing tree of the comment; 3) var-

ious grammatical tags such as part-of-speech (e.g. Verbal Phrase and Noun Phrase) and Named-Entity Recognition (NER) labels (e.g. Person, Location, and Time). This kind of structured personal information is stored and used later in various ways: e.g. to improve personal search services by retrieving results that are relevant to the named-entities related to the user.

Thanks to the growing amount of personal data that are available to be collected, it is easy to predict that personalized services will continue to evolve and expand. There are, however, limitations and concerns. First, the current methods of information extraction are not feasible for any small company that doesn't have its own email service because they are based on accessing large data sets collected with proprietary email services. Second, keeping large amount of structured personal information on centralized remote cloud servers raises privacy and security concerns [10, 46].

To address these problems, we present LN-Annote (Locally-customized NER-based Annotation), a novel information-extraction subsystem that is designed and optimized to process the email data available *locally* on each personal mobile device. Our contributions include:

- a distributed learning model based on two phases: *universal training* to generate a common parameter set on the cloud and *custom training* to refine and optimize the shared common parameter set by using the email data locally available on each mobile device;
- a discussion on how to extend the architecture of a personal search system to integrate the LN-Annote subsystem;
- an implementation of LN-Annote using locally available information and optimization methods leveraging the GPU on the mobile device; and
- an extensive set of experimental results to prove the feasibility, effectiveness, and efficiency of our approach.

In Section 2 we describe a personal search service as an example of a personal information system where LN-Annote is employed as a subsystem. In particular, we compare two different approaches for information extraction, on the cloud and on the mobile. Also, we illustrate the workflow of LN-Annote to extract personal information from emails stored on smart devices, which can then be available for many personalized service providers. In Sections 3 and 4 we present how we implemented and optimized our system. In Section 5 we present a comprehensive set of experiment results to show the efficiency and the effectiveness of our approach. In particular, we show the advantages provided by the addition of custom training on top of universal training.

2. LN-Annote SYSTEM DESIGN

In this section we present the design of LN-Annote, its main components, and how these interact with other modules as part of a bigger system. LN-Annote is an NER-based system optimized to extract information from email messages, in particular those sent via SNSs. The focus on email is motivated by two observations: 1) email messages convey scads of personal information and 2) many web services send notification emails with very useful data such as reservations or recommendations.

2.1 Information Extraction and NER

Nowadays many companies send emails to their customers for various purposes such as discount offers, purchase history, appointment reminders, or activity updates on SNSs. As more companies integrate their services with email systems, these email messages contain a growing amount of personal information. Meanwhile, more and more people use their email to manage personal information [54]. Hence, the ability to extract personal information from emails becomes increasingly important. Nonetheless, existing extraction techniques have various limitations. One approach is to write vendor-specific parsing scripts; this, however, requires a large amount of manual labor to update the scripts whenever the vendor changes the email format. Another approach is to use Microdata embedded in the emails containing structured information [24]; this, however, is currently not very effective because the number of email messages that contain Microdata is very limited.

To overcome these problems, Natural Language Processing (NLP) techniques have been proposed to assist service providers in extracting useful information [37, 50]. Named-Entity Recognition (NER) is a popular NLP technique to classify given vocabularies into predefined categories [9, 36]. A wide variety of systems use NER for different text types such as queries, SNS posts, or résumés [23, 29, 38].

The performance of a NER system can be evaluated using various metrics. One of the most widely used metric is the F_1 *score* which is defined as the harmonic mean of *Precision* and *Recall*:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (1)$$

Precision, or *Positive Predictive Value*, is the correctness of the predicted classification and Recall, or *True Positive Rate*, is the coverage of the positive cases.

2.2 A Use Case: Personal Search Service

The service infrastructure that we present here is an example of a system where LN-Annote works as a key component subsystem in collaboration with other components.

A personalized search service provides answers based on the personal information that it has collected from the emails of the user who requested the query. The personal information of each user is collected periodically from the email database and stored into a structured database to simplify its retrieval. The diagrams of Fig. 2 illustrate two different approaches to implement this service. Each shaped object represents a processing component or a document database and each arrow indicates a direction of data flow. The users access the services through their smart devices, searching personal information through a *Personal Search App* and receiving emails through an *Email App*.

Fig. 2(a) shows an approach where the extraction happens in the cloud. Periodically, e.g. once a day, the extraction system accesses directly the email database to update the *Structured Personal Information Database*. When a user issues a personal query with the *Personal Search App*, the query is passed to the personal search where it is handled by the *Query Processor* to disambiguate it and augment it. The processed query is then passed to the *Personal Information Search Engine* which retrieves relevant documents from the Structured Personal Information Database. The retrieved documents are ranked by the *Search Result Ranking* system and returned to the particular app running on the mobile device, e.g. the Personal Search App, so that the user can see the results. This approach, however, has several disadvantages. First, it is feasible only for a very small group of service providers that own email services with a sufficient number of users. Hence, many service providers that do not

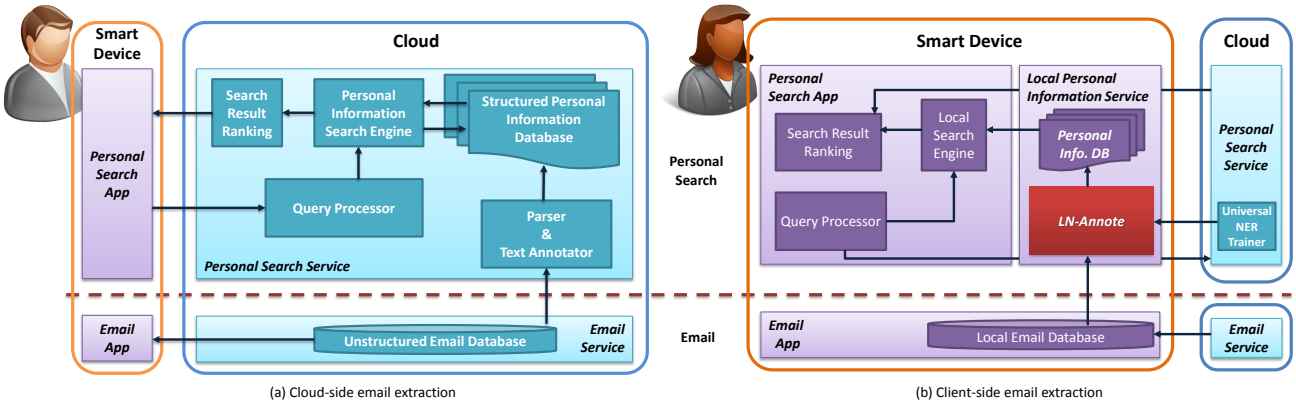


Fig. 2: An architectural comparison between two personal search systems.

have access to email services miss a major source of personal information. Second, it raises privacy and security concerns over the extraction, storage, and processing of very large amount of personal information in centralized remote cloud servers [28].

To address these challenges we propose the approach illustrated in Fig. 2(b). In this approach, the personal information extraction becomes a task that runs locally on the personal device of each user, where recently fetched emails are stored by the email app.[1] Our proposed LN-Annote in *Local Personal Information Service* is similar to the *Parser & Test Annotator* of Fig. 2(a) but uses NER and creates *Personal Information Database* on the local device. As shown in Fig 2(b) the local extraction of personal information resolves the dependency between the personalized service and the email service, thus allowing personalized service providers without their own emails services to access the local personal information database. Also, since the information remains local in the mobile device, privacy concerns and security issues are effectively alleviated [49]. Finally, the introduction of LN-Annote, a distributed NER subsystem, improves considerably the extraction accuracy while reducing the computation burden on the cloud servers (as discussed in more detail in Section 6). On the other hand, our approach requires that the mobile devices perform some additional amount of computation and this may have a negative impact on their overall performance and energy consumption. To minimize this impact, we have developed a method for custom training based on feature templates (as described in Section 4.1) and we have parallelized the key algorithms to run on the GPU present in each modern mobile device (Section 4.2).

2.3 The LN-Annote System Workflow

In this section, we describe the LN-Annote subsystem that we built to implement the approach shown in Fig. 2(b). To achieve more accurate prediction, we conceived a novel method that performs training for NER in two separate main steps, as illustrated in Fig. 3 (in this flowchart, a solid arrow represents a flow of data and a dashed arrow represents a sampling activity.) The first step, shown in the blue box (left), is *universal training*: this is essentially identical to traditional learning and returns learning parameters that will be **shared among all users**. The second step, shown in the red boxes (right), is *custom training*: it runs on each personal device, takes the learning parameters from the universal training, and enhances them by further training with the **locally accessible dataset which is specific to each user**. The main goal of our method is to produce locally-customized learning parameters that work well for the particular local environment. However, the parameters need to perform well also on the global texts, by preserving the knowledge from universal training.

As shown later, LN-Annote achieves extraction performance comparable to training for a combination of the global dataset and the personal dataset, while requiring a significantly smaller amount of computation. Next, we provide more details on the two main steps.

1. Universal Training in the Cloud. This step consists of two substeps:

1-a. *Data sampling and NER labeling* is a preparation activity that takes samples from a universal text database and creates labels for the sampled data to feed the supervised training of Substep 1-b. Choosing a representative dataset with an appropriate amount is an important task for the quality of the training [34]. The sampled data can be labeled using different methods: a) manual labeling, b) manual labeling and running semi-supervised learning, and c) running unsupervised learning [17, 51]. For the experiments of this paper we used the CoNLL03 dataset provided with manually labeled NER tags [47], while semi-supervised learning is commonly used for large datasets.

1-b. *NER universal training* uses supervised learning to process the labeled data produced by Substep 1-a. In traditional machine learning, the learning parameters created by this kind of algorithm are directly used to test the prediction of NER labels for the actual dataset. In our approach, instead, these learning parameters are shared to the multiple mobile devices for custom training.

2. Custom Training & Testing on the Mobile Device. This step consists of three substeps:

2-a. *Data sampling & semi-supervised NER labeling* works similarly to Substep 1-a to produce labeled data for Substep 2-b. Here, the inputs are text samples selected from the local email database used by the email app running on the mobile device. Also, in this case the manual labeling cannot be applied because it is infeasible to ask the user of the personal device to do it. Instead, we obtained *local gazetteers*, a list of named-entities from a reliable source [33]. This substep is performed automatically by using a semi-supervised learning

[1] Accessing other app's database is not trivial on smart platforms where each app runs in its own sandbox environment, but it is still possible through a couple of options. We have implemented email apps that share email data with other allowed apps through inter-app communication methods allowed in each platform, e.g. *Content-Provider* on Android [2]. The users can control which apps are allowed to access the emails.

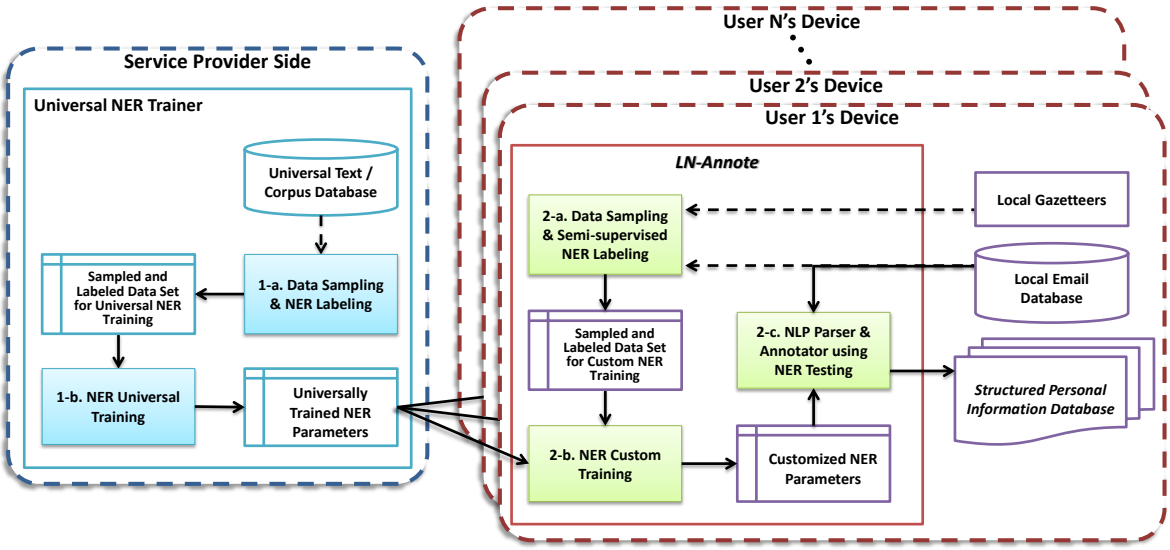


Fig. 3: The flowchart of LN-Annote.

algorithm based on the labels from the gazetteers. How to obtain gazetteers is explained in Section 3.3.

2-b. *NER Custom Training* updates the NER parameters by learning from the labeled dataset generated by the emails on the mobile device. The use of updated parameters is expected to keep the same performance as the use of universally trained parameters on the global dataset, while delivering better performance on the local emails.

2-c. *NLP Parser & Annotator using NER*, the final substep of LN-Annote processes the emails on the mobile device. This is done using NER based on the parameters created in the Substep 2-b. The outcome is stored in the Structured Personal Information Database where it can be used by any personal services running on the mobile device, as long as this is allowed by the user.

Notice that each substep of LN-Annote occurs with a different frequency: universal training (Substeps 1-a and 1-b) is done only once on the cloud servers; custom training (Substeps 2-a and 2-b), i.e. obtaining gazetteers, takes place periodically, e.g. once a week or a month; and finally, the actual information extraction (Substep 2-c) runs rather frequently, e.g. everyday.

3. NER SYSTEM IMPLEMENTATION

We designed the LN-Annote system to work with different types of incremental learning algorithms [27]. For instance, it can work as a framework running the universal training in the cloud and the custom training on the mobile while sharing the learning parameters between the two. In this section, we introduce as a showcase our *Neural Network* model for NER.

3.1 NER Algorithm Selection

NER systems may adopt a linguistic grammar-based model or a statistical learning model. We developed a system using a window-based neural network model for NER [13, 19]. Neural network learning algorithms learn the representation of multiple layers with increasing complexity and abstraction. A neural network works iteratively through *feed-forwarding*, a process to obtain the hypothesis and the cost objective (error), and *back-propagation*, a process to adjust the parameters according to the error.

Compared to various alternatives, neural network models are known to be difficult to understand with respect to the internals of the parameter optimization process. For this reasons, in many state-of-the-art NER systems programmers use other machine learning algorithms such as Conditional Random Fields [19, 52], Maximum-Entropy Markov Model [20], or Perceptron [12], which in many cases achieve pretty high F_1 score (over 90). Still, we chose to build a neural network model for NER for the following reasons. First, a neural network automatically chooses feature values through feed-forwarding and back-propagation. This makes the system free from the need of using carefully hand-crafted features that are required by other learning techniques. In particular, in our system it enables the automation of the preparation and training for the customized learning step. Second, neural network algorithms make heavy use of linear algebra and, particular, matrix operations: these can be easily parallelized and executed on the GPU of the mobile device, thereby reducing the time and energy spent on the computation in the custom training.

Neural network models are prone to *catastrophic forgetting* (or catastrophic interference), an extensive loss of previously learned information that may occur while learning new training data. This problem has its main causes in the representational overlap within the learning system and in the side-effects of the training methods used (rather than in the lack of sufficient computing resources in the system) [15]. In our system, catastrophic forgetting could manifest in a particular way: the customized NER parameters can recognize entities learned during custom training while losing the ability of identifying entities previously learned during universal training. There are several known solutions to mitigate catastrophic forgetting, including: conservative training and support vector rehearsal [8], multi-objective learning [26], sweep rehearsal [43], unsupervised neuron selection [22], fixed expansion layer [14], and ensemble of classifiers [40]. In our neural network model we avoid catastrophic forgetting by designing an incremental neural network model that introduces a scale factor to the weight adjustment [21, 53]. This leads to excellent results, as discussed in Section 5.

3.2 A Neural Network Model for NER

Both universal training and custom training are based on supervised learning and use the same neural network model. In fact, the two training algorithms are essentially the same

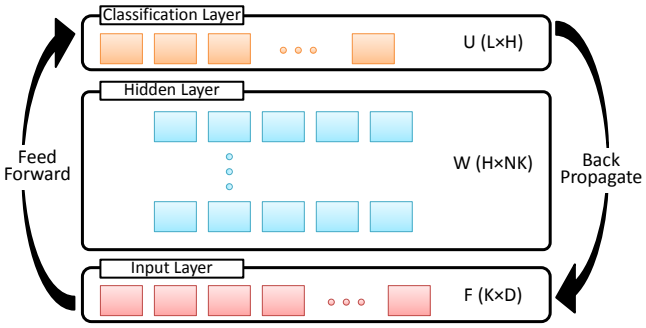


Fig. 4: The designed neural network architecture.

for consistency and compatibility of network parameters. The only differences are the number of iterations and the learning rate. We use a window-based network model with a fixed window size. This means that, while labeling a given word, a few nearby words are also processed to provide a local context that helps capturing the characteristics of the target word.

In the neural network we have three layers, as shown in Fig. 4: the input layer, the hidden layer, and the classification (output) layer. The input layer consists of feature vectors, or word representations, obtained from the unsupervised pre-training [25]. Each feature vector consists of K float values to represent a vocabulary in the dictionary of D vocabularies. The hidden layer, with a dimension of H, is used to derive hidden features by computing the input layer and the weight vectors. The dimension of weight vectors W for the hidden layer is $H \times NK$ where N is the size of window. On top of the hidden layer, there is the output layer for *softmax regression* to predict named-entities [16]. The weight vectors U for the output layer is of size $L \times H$, where L is the number of possible output classes.

Feed-Forwarding. Feed-forwarding in neural networks is a process to compute prediction values. We first calculate $z_i = Wx_i + b_1$ where x_i is the feature vectors of the words in the i-th window and b_1 is a bias vector. Then, we obtain $a_i = f(z_i)$ by applying a nonlinearity function f. Finally, we compute the hypothesis $h_i = g(U^T a_i + b_2)$, where b_2 is a bias for the softmax regression; the sigmoid function g makes the regression result fit smoothly into a classification result. The final prediction, h_i, is the probability that x_i be a named-entity of each class. Here, along with the feature vectors used as the input layer, the weight vectors and the bias vectors are considered as the NER parameters of Fig. 3.

Algorithm Configuration. We use the hyperbolic tangent function *tanh* as the nonlinearity function f. Note that its derivative can be expressed with the *tanh* function itself, i.e. $f'(x) = \frac{d}{dx}\tanh x = 1 - \tanh^2 x$. This greatly simplifies the back-propagation, thereby reducing the amount of computation for training and testing on the mobile side. The sigmoid function we use is $g(z) = \frac{1}{1-e^{-z}}$.

Training Objective. In both the universal training and the local training we minimize the same cost (objective) function which is the following cross-entropy error with a regularization factor:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{L}\left[1\{y^{(i)} = j\}\log\left(h_\theta^{(i)}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$$

Platform	API
Android	*ContactsContract.Contacts.CONTENT_URI*
iOS	*ABAddressBookCopyArrayOfAllPeople*
Windows Phone	*Microsoft.Phone.UserData.Contact.SearchAsync*

Table 1: Mobile APIs for *PER* entities.

Provider (Base URI)	REST API (PER) (ORG/LOC/MISC)
Facebook	*/v2.1/me/friendlists*
http://graph.facebook.com	*/v2.1/me/feed?with=location*
Google+	*/v1/people/me/people/connected*
https://www.googleapis.com/plus	*/v1/people/me/activities/public*
Twitter	*/1.1/followers/ids.json*
https://api.twitter.com	*/1.1/statuses/user_timeline.json*

Table 2: SNS open APIs for various entities.

where the model parameter θ includes the input values extracted by the current window, λ is a weight decay term (for any $\lambda > 0$) introduced for regularization, and $1\{condition\}$ is a function that returns 1 when *condition* is *true* and 0 otherwise. During back-propagation, we minimize this objective function using stochastic gradient descent, a first-order optimization method widely used for training a variety of models in machine learning.

3.3 Local Gazetteer from Mobile

Differently from universal training, which can benefit from the manually tagged corpus, the custom training on mobile devices cannot rely on any human effort to label the dataset. To automate the custom training while obtaining a high-quality training dataset, we used the idea of gazetteers [33] and developed a method to acquire gazetteers from local and external sources. Gazetteers are a named-entity list obtained from an external, reliable source. Our gazetteer-induction method uses the contact list, checked-in locations, and liked pages through the mobile platform and SNS open APIs. The available APIs for collecting this gazetteer information from mobile and SNS platforms are listed in Table 1 and Table 2, respectively. The labeled dataset is then fed into the semi-supervised learning algorithm to create a training set for the custom training.

4. OPTIMIZATION

The execution of custom training on the personal mobile device poses some challenges in terms of increased energy consumption and the possible slowdown of the overall device. To address these challenges we performed two major optimizations.

4.1 Feature Templates

Our neural network model for custom training is designed for enhancing accuracy by using the information available on the local device. Hence, it is important to capture the peculiar vocabularies observed during the custom training on the device. For example, non-standard words (frequently used on the web or in text messages as shown in Fig. 1) or abbreviated words (such as 'CROACC' [2]) are less likely to be part of a general dictionary and, therefore, less likely to have occurred during universal training. They, however, may occur frequently in the custom training with the database of a particular user. If so, for this user these words may be

[2]Cannot Rule Out Anything, Correlate Clinically.

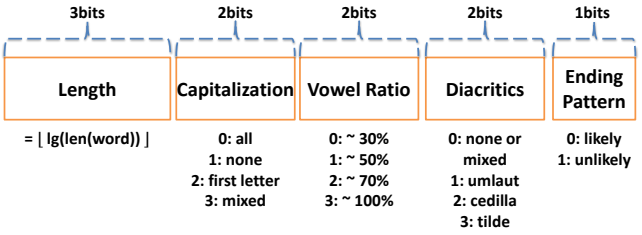


Fig. 5: The index of grouped feature templates.

named-entities or may be useful to determine nearby named-entities.[3] Thus, we capture the newly discovered words and add them to our vocabulary list, while also creating feature vectors for these new words. A performance concern, however, arises if we initialize the new feature vectors with random values. We keep the learning rate low in the custom training because it is additional training on top of the universal training and the scale factor of incremental back-propagation limits the learning rate to prevent catastrophic forgetting. Starting from parameters with the random values require more learning iterations, which slows down the training. To address this challenge, we developed a new technique to assign initial values to the feature vectors for the newly discovered words in the custom training. This technique copies template feature vectors from a group to which a new word belongs. Then, further training can converge with less learning iterations while keeping the learning rate low. As shown in Fig. 5, we use local features for grouping each word [30, 48]. The *Ending Pattern* is a boolean value representing if the word ends with one of the predefined postfixes, such as "ie", "son", "ng", or "a". These local features are then encoded to calculate the template group index for feature vectors, with a maximum index of $2^{10}-1=1023$. For instance, the group index of a new word (*YoungHoon*) is 488. Then, the feature vectors for the word are copied from the template at index 488.

4.2 Hardware Acceleration

While feature templates can reduce the number of iterations to learn new vocabularies, the largest performance bottleneck still lies in the training algorithm. This leads into problems including: high energy consumption, CPU occupation, and delay in using the training results. In previous works, NER algorithms were accelerated by using some hardware units available on computer servers [18, 35]. Our goal, however, is to remove the performance bottleneck during the custom training on the mobile devices.[4] Furthermore, mobile devices are particularly prone to these problems due to their relatively slower CPUs and more limited energy budget. To accelerate the NER custom training and testing algorithm we exploit the GPU on the mobile device. Besides performance gains, GPU-based acceleration can also lower energy consumption due to the decreased execution time compared to the execution on the device CPU.

The implemented OpenCL kernel executes the entire training algorithm, e.g. window index extraction, hypothesis computation, and gradients calculation. Listing 1 shows an OpenCL kernel that computes the hypothesis of the softmax regression (the top layer in Fig. 4) for a NER class.[5]

[3]Notice that this is different from the case in universal training when newly discovered training vocabularies that have no corresponding feature vectors are discarded because they appear rarely.

[4]In most cases, training requires more computation than testing for the same size of dataset.

[5]This implementation is from our second version where each kernel computes a small portion of the algorithm.

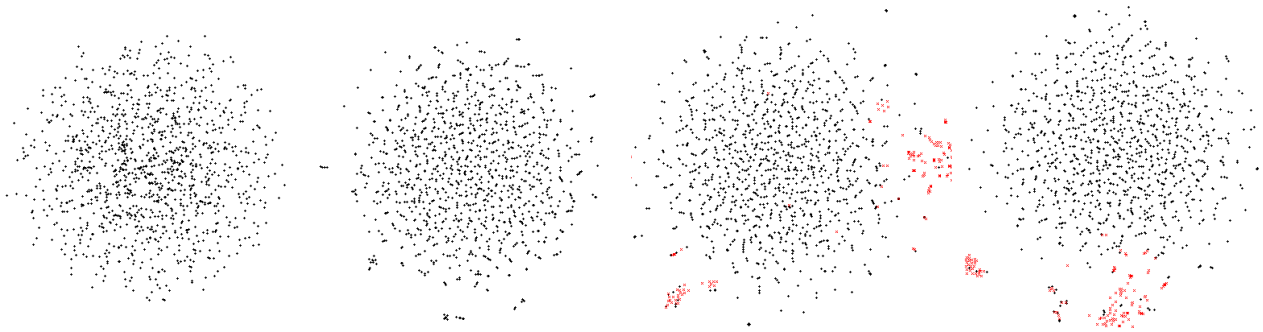
```
1  /* Put product of global values to local mem */
   if (gid * 4 < hiddenSize)
3    partial_dot[lid] = U[gid] * a[gid];
   else
5    partial_dot[lid] = 0;
   barrier(CLK_LOCAL_MEM_FENCE);
7
   /* Repeatedly add values in local memory */
9  int nElem = group_size;
   int i = (nElem + 1) / 2;
11 do {
     if(lid < i && lid + i < nElem) {
13     partial_dot[lid] += partial_dot[lid + i];
     }
15   barrier(CLK_LOCAL_MEM_FENCE);
     nElem = i;
17   i = (i + 1) / 2;
   } while (i != nElem);
19
   /* Transfer final result to global memory */
21 if(lid == 0) {
     h[get_group_id(0)]
23     = dot(partial_dot[0], (float4)(1.0f));
   }
25
   barrier(CLK_GLOBAL_MEM_FENCE);
27 if (gid == 0) {
     for (int i = 1; i < get_num_groups(0); i++)
29     h[0] += h[i];
     h[0] += b2;
31   h[0] = 1.0 + pow((float) M_E_F, -h[0]);
     h[0] = 1.0 / h[0];
33 }
```

Listing 1: A portion of OpenCL kernel that computes the hypothesis.

This kernel implementation uses a technique that first calculates the dot products of sub-matrices on the local memory storage shared among GPU thread blocks and then it sums up these partial dot products to quickly compute the dot product of the entire matrix. This exploitation of the local memory speeds up the summation process because for a GPU core accessing the local memory is faster than the global memory [55].

5. EXPERIMENTS

In the following experiments, we used the CoNLL03 shared task [47] in the universal training while we leveraged emails dataset in the custom training. CoNLL03 provides an English training dataset (*CoNLL03_train*) and two English test datasets (*CoNLL03_testa* and *CoNLL03_testb*). The actual email datasets used in the experiments were created with SNS notification emails, such as the example presented in Fig. 1(a), chosen from personally donated emails for research purposes. We created a training dataset and a testing dataset for each user and used these in Substeps 2-b and 2-c of Fig. 3, respectively. Let *email_traina* denote the training dataset from user_a and *email_testa* denote the testing dataset from the same user. While a semi-supervised learning and gazetteers were used with the email datasets for training as explained in Section 2.3, the email datasets for testing were labeled following the CoNLL03 guidelines [47] for the experiments in this section. Each email dataset for custom training consists of 128,000 words out of which approximately 8,000 to 21,000 words are unique within that dataset. The used email dataset for testing contains around 64,000 words. Each measured value that has a possibility of variation, such as execution time and power consumption, was executed 10 times and averaged excluding the largest observation and the smallest observation.



(a) Random (b) Universal train on CoNLL03 (c) Custom train on email_traina (d) Custom train on email_trainb

Fig. 6: t-SNE plots of feature vectors.

	Training			Testing		
	Prec.	Recall	$F_{\beta=1}$	Prec.	Recall	$F_{\beta=1}$
LOC	84.28%	80.02%	82.10	83.15%	75.87%	79.34
MISC	79.24%	72.45%	75.69	73.86%	72.34%	73.10
ORG	85.58%	84.56%	85.07	78.64%	75.24%	76.91
PER	89.44%	86.43%	87.91	87.78%	83.44%	85.56
Overall	85.46%	81.88%	83.63	82.06%	77.54%	79.74

Table 3: CoNLL03 evaluation of universal training.

	Training			Testing		
	Prec.	Recall	$F_{\beta=1}$	Prec.	Recall	$F_{\beta=1}$
LOC	83.83%	80.58%	82.17	83.00%	77.27%	80.03
MISC	79.14%	72.02%	75.41	72.31%	69.96%	71.11
ORG	85.56%	84.41%	84.98	78.08%	75.17%	76.60
PER	89.28%	86.32%	87.77	87.21%	82.90%	85.00
Overall	85.25%	81.91%	83.55	81.48%	77.41%	79.39

Table 4: CoNLL03 evaluation of custom training.

5.1 Learning NER Feature Vectors

In this section, we compare how the feature vectors, the main learning parameter in our NER neural network model, change as we execute universal training and custom training.

One of the difficulties in using a neural network model is understanding its internal behavior. In many cases, visualizing how the learning iterations progress can help developers understand and improve the system. t-Distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear method to transform data with multiple dimensions into data with reduced dimensions. This technique is particularly popular for visualizing high-dimensional data on a two- or three-dimension plot. The idea of t-SNE is to minimize the difference between the two distance matrices across each point on the original space and the reduced space.

The diagrams in Fig. 6 visualize the feature vectors to investigate how they change through the universal training and the custom training. In these diagrams, each dot represents a word. A black dot is a word that was present in the initial vocabulary set. A red dot is a word newly encountered during custom training. Fig. 6(a) shows the randomly initialized values in the feature vectors before the pre-training. The values are mostly scattered in the two dimensional space without forming any specific patterns, with a few areas which are denser. The values in Fig. 6(b) are generated from the feature vectors which were originally copied from the pre-trained feature vectors and then universally trained on *CoNLL_train*. Here, the dots are spread around more than in the case of the random initial vector. We believe that across the many iterations of the training the values in each word representation have moved to establish a uniform distance from one another. Meanwhile, we can observe a new pattern: a few dots form short lines. This linear pattern can be also spotted in the following two figures. Fig. 6(c) shows the feature vectors obtained by copying the universally trained feature vectors in Fig. 6(b) and updating them by training on *email_traina*. The feature vectors in Fig. 6(d) are generated in the same way but trained with *email_trainb*. While these two figures maintain the forms and patterns of Fig. 6(b), the newly introduced red dots tend to stand close to some other red dots, thus forming a group. These groups likely incorporate named-entities that never existed in the initial vocabularies and that are very relevant to a particular user, such as the name of a friend or a local restaurant.

5.2 NER Performance Comparison

In LN-Annote, the universal training is done with supervised learning on the CoNLL03 training (or the *dev*) dataset. Table 3 shows the general NER performance of universal training. In particular, the left half of the table is tested with *CoNLL03_train* and the right half is tested with *CoNLL03_testa*. The results show that in general Precision is higher than Recall for every entity type and, based on Equation 1, the F_1 scores remain in the range $[70, 90]$. Obviously, all the NER performance results are a bit lower on the testing dataset than on the training dataset because the parameters used in this experiment are originally learned from the training dataset.

Meanwhile, Table 4 presents the NER performance of custom training. As described in Section 2.3, the parameters used in this experiment are the outcome of custom training, which takes as input the learning parameters from universal training. Then, the custom training further learns from a local email dataset (*email_traina*), updating the parameters. Like for Table 3, the results of Table 4 were tested on the CoNLL03 training (left) and testing dataset (right). Therefore, comparing the tables shows how much (negative) impact, such as catastrophic forgetting, was introduced in learning by custom training.

The results show that most values remain the same, with a slight decrease compared to the values in Table 3. Hence, catastrophic forgetting is avoided. Note that on both the training and testing datasets the recall value of *Location* (LOC) is higher (80.58 and 77.27) than before (80.02 and 75.87). This possibly means that the custom training with the local email dataset improves the parameters so that the algorithm can better recognize Location entities. For instance, there is no sentence that contains the two consecutive words “on Moscow” in the training dataset while the

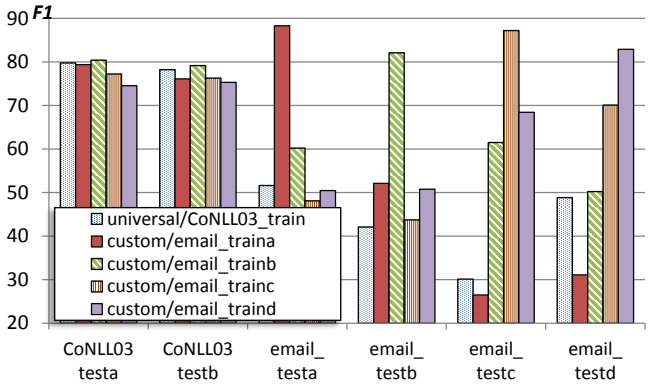


Fig. 7: F_1 scores of universal and custom training.

Platform	API
Model	Moto G
Chipset	Qualcomm MSM8226 Snapdragon 400
CPU	Quad-core 1.2GHz Cortex-A7 (Krait)
GPU	Adreno305 400Mhz
Main Memory	1GB
GPU Memory	441MB
Flash Storage	16GB
LCD Resolution	720 x 1280
Base OS	Linux-3.4.42 (LTS)
Platform	Android 4.4.4
OpenCL	Embedded Profile v1.1

Table 5: Device Specification.

testing dataset has it. The NER algorithm with the universally trained parameters tagged "Moscow" in that sentence in the CoNLL03 test dataset as *O* (non-entity). On the other hand, customized parameters which learned from a local email dataset made the NER algorithm tag "Moscow" in that sentence as *I-LOC* (Location). This implies that the custom training can improve the performance not only on the specific dataset for which it is trained but also on the general dataset.

5.3 Cross Evaluation of Learning

In this experiment, we use five parameter sets including: the one universal-trained with *CoNLL_train* and four parameter sets custom-trained with *email_train[a-d]*. The bars in Fig. 7 indicate evaluated values with different training and testing datasets. The evaluations are grouped by the six testing datasets: *CoNLL_test[ab]* and *email_test[a-d]*. For example, the leftmost bar indicates the F_1 score obtained from testing *CoNLL03_testa* with the parameters learned from *CoNLL03_train*. Likewise, the rightmost bar is obtained by testing *email_testd* on the parameters trained with *email_traind*. The first two groups present similar F_1 scores, ranging from 74 to 81. This shows that all these parameters can stably recognize entities in *CoNLL_train* even after custom training. In the next four groups, the customized parameters show a strong F_1 on the test datasets from the same user's emails on which the customized parameters were trained. In fact, in real systems a parameter set trained on a user's emails will never be used on a different user's emails. However, these evaluations across different custom datasets are conducted to analyze how custom training affects the performance of NER on different personal datasets. An interesting point is that the two bars, *email_testc* on *email_traind* and *email_testd* on *email_trainc*, have higher F_1 scores than other cross evaluations, e.g. *email_testb* on *email_trainc*. We speculate that this can happen when the

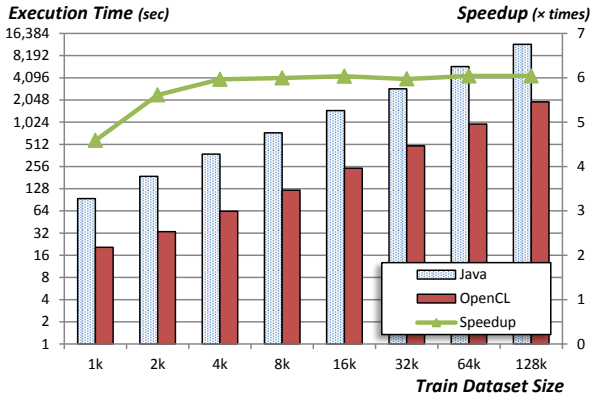


Fig. 8: Execution time comparison (H=64).

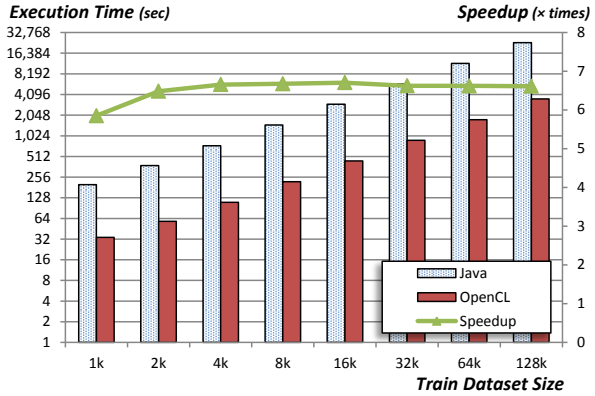


Fig. 9: Execution time comparison (H=128).

two users share many named-entities, e.g. by having common friends or living close to one another. Similarly, the cross evaluation tends to have a certain level of correlation between the two customized parameter sets. This, however, does not mean that the relationship is always symmetric. For instance, the evaluation of *email_testb* with a parameter set from *email_trainc* has a low F_1 of 43.73 while the evaluation of *email_testc* with *email_trainb* has 61.5, close to the average cross evaluation score.

5.4 OpenCL Performance Speedup

In this section, we evaluate our efforts on executing the NER training algorithm to relieve the increased CPU occupation and power consumption caused from having more computations on the mobile devices. The experiments are done on a low-end Android phone. The detailed specification of the tested phone is listed in Table 5.

Fig. 8—10 compare the execution time of custom training by using the CPU and the GPU on the mobile device. The white bars represent the execution time of the CPU implementation written in Java. The red bars show the execution time of the GPU implementation written in OpenCL and embedded in the Java application through the Java Native Interface (JNI). The green curves indicate the speedup achieved by the OpenCL implementation (ran on the GPU) over the Java implementation (ran on the CPU).

Fig. 8 shows the execution time of the training algorithm when the hidden layer size *H* is set to 64. The execution times of both Java and OpenCL implementations grow proportional to the input size, i.e. the number of vocabularies in the training data set. In most cases, the speedup is close to 6, while the speedup is 4.5 and 5.5 for train dataset sized 1k and 2k, respectively. This interesting phenomenon occurs

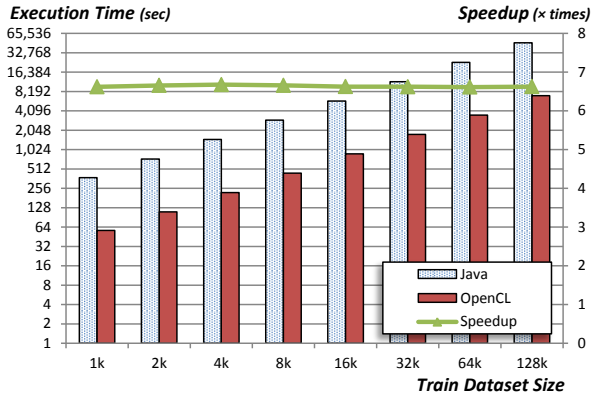


Fig. 10: Execution time comparison (H=256).

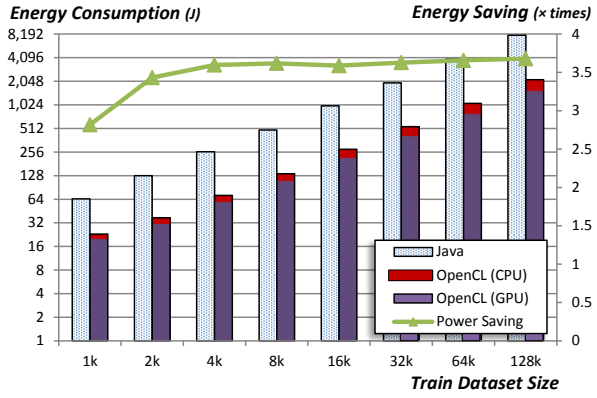


Fig. 11: Energy consumption comparison (H=64).

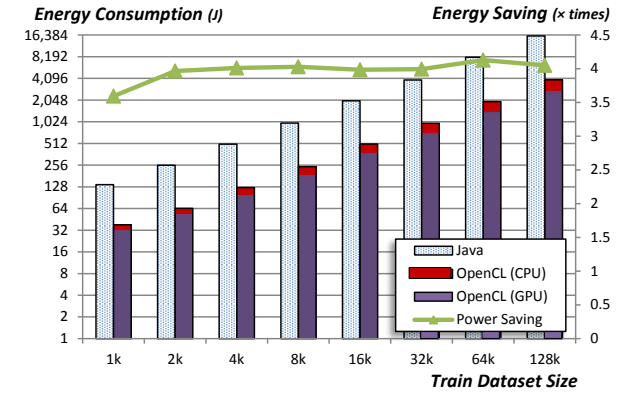


Fig. 12: Energy consumption comparison (H=128).

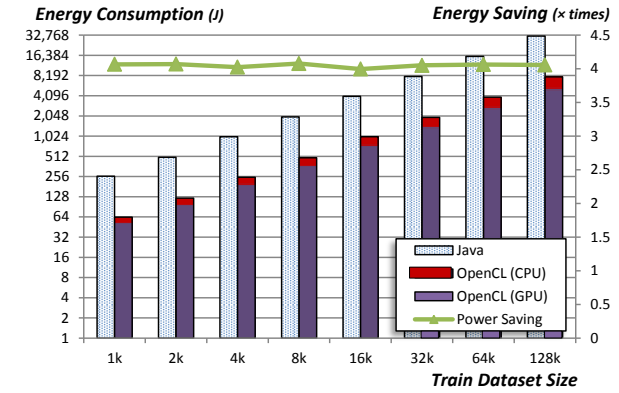


Fig. 13: Energy consumption comparison (H=256).

because for a small dataset the overhead from the OpenCL kernel invocation takes a large portion of the total execution time. This overhead includes the times for initializing the OpenCL context, compiling the OpenCL kernel, and copying kernel arguments.

Fig. 9 is the execution time when H is set to 128. While increasing H improves prediction performance, it takes more time to complete the computations. Compared to Fig. 8 the execution times are almost doubled. This means that the size of the hidden layer impacts the amount of computations proportionally. The lower points on the left end of the green curve are observed also in this figure. The bending slops is more gradual than in Fig. 8. The overall speedup from the previous figure is increased because the OpenCL performance gets better as H increases. This is because our OpenCL implementation exploits the benefit of the concurrent hardware threads, which execute the matrix operations parallelly.

Fig. 10 presents the execution time when H is 256. Since the amount of computations required in each iteration has increased because of the larger H value, the lower speedups observed in the previous figures do not appear in these experiments. The overall speedup remains the same without a large improvement with respect to the previous experiment.

5.5 OpenCL Energy Saving

Our next set of experiments is about energy consumption. Since we have achieved a good speedup by utilizing the mobile GPU, we also expect to see some reduction in energy consumption. To measure the power consumption of each application and component, we used an open source software called *PowerTutor* [56] and a profiling tool, called *Trepn*, provided by the chipset vendor [41].

Fig. 11 shows the energy consumed by the Java implementation executed on the CPU and by the OpenCL implementation run on the GPU. The red portion of the energy consumption of the OpenCL implementation is spent by the CPU while the rest is spent by the GPU. These experiments confirm that the energy consumed on the training for the same amount of input dataset could be reduced to one fourth by using the mobile GPU. This reduced energy consumption mostly came from the decreased execution time by the GPU. In fact, the power dissipation is even higher while the algorithm runs on the GPU. For instance, when the algorithm runs on the CPU the average power consumption by the CPU is approximately 0.682 Watt. On the other hand, the average power consumptions by the GPU and the CPU when the algorithm is executed on the GPU are around 1.02 Watt and 0.108 Watt, respectively. Another interesting point is that the power dissipation on the CPU, while the GPU is in use, is very low. This is because our OpenCL implementation is one consolidated kernel, thus not relying on the CPU during the entire iteration.[6] We speculate that the 0.108 Watt is mostly spent on the static power dissipation and on the maintenance of the Android app main thread, including event handling of the user interface.

Both Fig. 11 and Fig. 12 show lower energy savings for small datasets, i.e. 1k and 2k. Compared to the green curves in Fig. 8 and Fig. 9, the green curves in Fig. 11 and Fig. 12 stagger slightly. We presume that this is due to small errors introduced by the power measurement tools we used.

[6]There exist some GPUs that consume CPU resources even inside the GPU kernel, although the GPU we used in the experiments does not.

6. IMPACT ON MOBILE CLOUD SYSTEM DESIGN

One of the advantages of our approach is to decrease the computational burden in the cloud. Although the server computers in the cloud are faster than mobile devices by at least one order of magnitude, in some cases the large number of mobile devices hides the gap in computational power. This is important for the design of mobile cloud computing systems where many users access the cloud service with their own mobile devices. In this section, we discuss the design aspect of our extraction system by comparing two different approaches: 1) having the universal training and all the custom trainings on the cloud and 2) having the universal training on the cloud and the custom trainings on the mobile devices.

Let's assume that our system delivers a personalized service to ten million users who access it through their mobile devices. In general, the required number of computers depends largely on the characteristics of the service. The average number of users one server computer can handle for our service could fall in a range between 1,000 and 10,000. The lower bound, 1,000 users, is calculated as follows: according to the Amazon Web Services (AWS)'s Total Cost of Ownership (TCO) calculator, the cost for using 10,000 servers (thus making each server handle 1,000 users) is $9,224,460.[7] This would take a large portion, if not most, of the revenues that a service provider with ten million service users can make.[8] On the other hand, there are various factors that limit the processing scale of a server computer to stay under 10,000 concurrent users. One factor is the challenge imposed by the network connection capacity [39, 31].

Despite this considerable number of computers, performing all the custom training tasks only on the cloud will likely incur a large delay in the service preparation cycle. For instance, the previous experiment on custom training of 128,000 words with H=128 took 23301.93 seconds on the mobile CPU but only 3523.09 seconds on the mobile GPU, as shown in Fig. 9. The execution of the same training task on a server computer with an Intel Xeon E5-2690 CPU takes 2527.19 seconds. When each of 10,000 servers has 32 CPU cores, this will allow each CPU core to execute the custom training for 31.25 users. Suppose that we have a synchronous batch system (such as Hadoop) for the custom training where each CPU executes one custom training task for one user. Each user requests custom training everyday. The requests, however, arrive at a random time of the day. There are 32 synchronized execution time slots a day, thus making each slot be 2700 seconds. Note that this time slot is large enough for custom training on a cloud machine (2527.19 seconds) and 32 slots can sufficiently handle 31.25 users. However, when we assume that the user requests follow normal distribution across the 32 time slots, we get an average number of requests in each slot equal to 1.812. By applying *Little's law* [44], the average response time is derived as $1.812 * 2527.19 = 4579.27$ seconds. This is slower than the response time of 3523.09 seconds that is achieved with the training on the mobile devices, where there is no delay in response time for processing requests from other users. In other words, the training on the mobile devices may also lead to better performance. Finally, this approach will also reduce the computational burden on the cloud, thereby reducing the cloud cost.

[7]This cost includes 32 computing cores, 128 GB memory, and 16 TB storage space [3].

[8]The Average Revenue Per User (ARPU) varies across the companies, ranging from $1.21 (Facebook) to $12 (eBay) [32].

7. RELATED WORK

LN-Annote is related to some existing studies that focus on enhancing the model parameters for distinct cases. For instance, domain adaptation is an approach to transfer the feature vectors obtained on the source domain to the target domain [11]. In speech recognition, speaker adaptation is used to to improve the recognition performance by adapting the parameters of the acoustic models to better match the specific speaker's voice [45]. LN-Annote can be roughly categorized as distance supervision because the custom training uses the local contacts and SNS accounts to label the email datasets [42]. One of the characteristic differences between these approaches and LN-Annote is that LN-Annote uses information and computational capacity available on the local device. This addresses some privacy concerns and reduces the computational burden on the cloud as shown in Section 5 and 6.

8. CONCLUSIONS

We proposed and implemented Locally-customized NER-based Annotation (LN-Annote), a new method to extract personal information from emails stored locally on personal mobile devices. Our implementation is based on a newly-designed neural network model that works well for the two main phases of learning that characterize LN-Annote: universal training (performed in the cloud) and custom training & testing (performed on the mobile devices). We also developed two methods for optimizing the training of the neural network: one is based on the use of feature templates and the other on leveraging the GPUs that are present on the mobile devices. The experimental results show the feasibility and effectiveness of LN-Annote. In particular, they demonstrate how the use of custom trained parameters actually improves the performance of NER on the local email data without reducing its performance on the dataset used for universal training.

Acknowledgments

We gratefully thank Ethan Kim, Jinan Lou, and Younggyun Koh for their precious comments. This work is partially supported by the NSF (A#:1219001).

9. REFERENCES

[1] Amazon Product Ads: www.amazon.com/productads.
[2] Android Developers: developer.android.com.
[3] AWS Total Cost of Ownership Calculator awstcocalculator.com.
[4] Facebook Ads: www.facebook.com/ads.
[5] Google Adsense: www.google.com/adsense.
[6] Google Now: www.google.com/landing/now.
[7] Siri: www.apple.com/ios/siri.
[8] D. Albesano et al. Adaptation of artificial neural networks avoiding catastrophic forgetting. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 1554–1561, July 2006.
[9] H. Assal et al. Partnering enhanced-NLP with semantic analysis in support of information extraction. In *Proc. of the Int. Workshop on Ontology-Driven Software Engineering*, pages 9:1–9:7, Oct. 2010.
[10] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Comm. of ACM*, 28(10):1030–1044, Oct. 1985.
[11] L. Chiticariu et al. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proc. of the Conf. on Empirical Methods in Natural Language Processing*, pages 1002–1012, Oct. 2010.

[12] M. Collins. Discriminative training methods for Hidden Markov Models: Theory and experiments with Perceptron algorithms. In *Proc. of Conf. on Empirical Methods in NLP*, pages 1–8, July 2002.

[13] R. Collobert et al. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, Nov. 2011.

[14] R. Coop and I. Arel. Mitigation of catastrophic interference in neural networks using a fixed expansion layer. In *Proc. of the Int. Symp. on Circuits and Systems*, pages 726–729, Aug. 2012.

[15] R. Coop and I. Arel. Mitigation of catastrophic forgetting in recurrent neural networks using a Fixed Expansion Layer. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 1–7, Aug. 2013.

[16] K. Duan et al. Multi-category classification by soft-max combination of binary classifiers. In *Proc. of the Int. Conf. on Multiple Classifier Systems*, pages 125–134, June 2003.

[17] D. Erhan et al. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, Mar. 2010.

[18] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Neural acceleration for general-purpose approximate programs. In *Proc. of the Int. Symp. on Microarchitecture*, pages 449–460, Dec. 2012.

[19] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proc. of the Annual Meeting on Assoc. for Comp. Linguistics*, pages 363–370, June 2005.

[20] M. Fresko, B. Rosenfeld, and R. Feldman. A hybrid approach to NER by MEMM and manual rules. In *Proc. of the Int. Conf. on Info. and Knowledge Management*, pages 361–362, Oct. 2005.

[21] L. Fu, H.-H. Hsu, and J. C. Principe. Incremental backpropagation learning networks. *Trans. on Neural Network*, 7(3):757–761, May 1996.

[22] B. Goodrich and I. Arel. Unsupervised neuron selection for mitigating catastrophic forgetting in neural networks. In *Proc. of the Int. Symp. on Circuits and Systems*, pages 997–1000, Aug. 2014.

[23] J. Guo et al. Named entity recognition in query. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 267–274, July 2009.

[24] M. Heinrich and M. Gaedke. Data binding for standard-based web applications. In *Proc. of the Symp. on Applied Comp.*, pages 652–657, Mar. 2012.

[25] E. H. Huang et al. Improving word representations via global context and multiple word prototypes. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 873–882, July 2012.

[26] Y. Jin and B. Sendhoff. Alleviating catastrophic forgetting via multi-objective learning. In *Proc. of the Int. Joint Conf. on Neural Networks*, pages 3335–3342, July 2006.

[27] K.-H. Kim and S. Choi. Incremental learning to rank with partially-labeled data. In *Proc. of the Workshop on Web Search Click Data*, pages 20–27, Mar. 2009.

[28] A. Kobsa, B. P. Knijnenburg, and B. Livshits. Let's do it at my place instead?: Attitudinal and behavioral study of privacy in client-side personalization. In *Proc. of the Conf. on Human Factors in Comp. Sys.*, pages 81–90, Apr. 2014.

[29] C. Li et al. TwiNER: Named entity recognition in targeted twitter stream. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 721–730, Aug. 2012.

[30] Q. Li and H. Ji. Incremental joint extraction of entity mentions and relations. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 402–412, June 2014.

[31] D. Liu and R. Deters. The reverse C10K problem for server-side mashups. In *Workshops on Service-Oriented Computing*, volume 5472 of *Lecture Notes in Computer Science*, pages 166–177. 2009.

[32] P. R. L. Monica. 5 reasons to not 'like' Facebook's IPO. *CNN Money*, pages 1–1, May 2012.

[33] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Proc. of the Int. Conf. on Advances in Artificial Intelligence*, pages 266–277, Dec. 2006.

[34] H. M. Nguyen et al. An alternative approach to avoid overfitting for surrogate models. In *Proc. of the Winter Sim. Conf.*, pages 2765–2776, Dec. 2011.

[35] E. Ordoñez Cardenas and R. d. J. Romero-Troncoso. MLP neural network and on-line backpropagation learning implementation in a low-cost FPGA. In *Proc. of the Symp. on VLSI*, pages 333–338, May 2008.

[36] S. Orlando, F. Pizzolon, and G. Tolomei. SEED: A framework for extracting social events from press news. In *Proc. of the Int. Conf. on World Wide Web Companion*, pages 1285–1294, May 2013.

[37] W. Paik et al. Applying natural language processing (NLP) based metadata extraction to automatically acquire user preferences. In *Proc. of the Int. Conf. on Knowledge Capture*, pages 116–122, Oct. 2001.

[38] S. Pawar, R. Srivastava, and G. K. Palshikar. Automatic gazette creation for named entity recognition and application to resume processing. In *Proc. of the Conf. on Intelligent & Scalable Syst. Tech.*, pages 15:1–15:7, Jan. 2012.

[39] A. Pintus, D. Carboni, and A. Piras. The anatomy of a large scale social web for internet enabled objects. In *Proc. of the Int. Workshop on Web of Things*, pages 6:1–6:6, June 2011.

[40] R. Polikar et al. Learn++: an incremental learning algorithm for supervised neural networks. *Trans. on Syst., Man, and Cybernetics*, 31(4):497–508, Nov. 2001.

[41] Qualcomm Technologies Inc. Trepn: Profiler starter edition user guide. In *Qualcomm Developer Network*, pages 1–46. Apr. 2014.

[42] A. Ritter et al. Modeling missing data in distant supervision for information extraction. *Trans. of the Association for Computational Linguistics*, pages 367–378, 2013.

[43] A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Proc. of the Int. Conf. on Artificial Neural Networks and Expert Syst.*, pages 65–68, Nov. 1993.

[44] K. Rust. Using little's law to estimate cycle time and cost. In *Proc. of the Conf. on Winter Sim.*, pages 2223–2228, Dec. 2008.

[45] G. Saon et al. Speaker adaptation of neural network acoustic models using i-vectors. In *Proc. of the Workshop on Automatic Speech Recognition and Understanding*, pages 55–59, Dec. 2013.

[46] E. Sarigol, D. Garcia, and F. Schweitzer. Online privacy as a collective phenomenon. In *Proc. of the Conf. on Online Social Networks*, pages 95–106, Oct. 2014.

[47] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of the Conf. on Natural Lang. Learning*, pages 142–147, May 2003.

[48] M. Tkachenko and A. Simanovsky. Named entity recognition: exploring features. In *Proc. of NLP*, pages 118–127, Sept. 2012.

[49] V. Toubiana, V. Verdot, and B. Christophe. Cookie-based privacy issues on Google services. In *Proc. of the Conf. on Data and App. Security and Privacy*, pages 141–148, Feb. 2012.

[50] J. Tsujii. Generic NLP technologies: Language, knowledge and information extraction. In *Proc. of the Annual Meeting on Assoc. for Comp. Linguistics*, pages 12–22, Oct. 2000.

[51] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 384–394, July 2010.

[52] S. N. V, P. Mitra, and S. K. Ghosh. Conditional random field based named entity recognition in geological text. *Int. Journal of Comp. Applications*, pages 119–122, 2010.

[53] J. H. Wang and H. Y. Wang. Incremental neural network construction for text classification. In *Proc. of the Int. Symp. on Comp., Consumer and Control*, pages 970–973, June 2014.

[54] S. Whittaker, V. Bellotti, and J. Gwizdka. Email in personal information management. *Comm. of the ACM*, 49(1):68–73, Jan. 2006.

[55] Y. Yang et al. Shared memory multiplexing: A novel way to improve GPGPU throughput. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 283–292, Sept. 2012.

[56] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of the Int. Conf. on Hardware/Software Codesign and Syst. Synthesis*, pages 105–114, Oct. 2010.