Sensei: Self-Supervised Sensor Name Segmentation

Jiaman Wu, Dezhi Hong, Rajesh K. Gupta, Jingbo Shang
Department of Computer Science and Engineering, University of California San Diego, CA, USA
Halıcıoğlu Data Science Institute, University of California San Diego, CA, USA
{j4wu, dehong, gupta, jshang}@eng.ucsd.edu

Abstract

Sensor names as alphanumeric strings typically encode their key contextual information such as their function or physical location. We focus here on sensors used in smart building applications. In these applications, sensor names are curated in a building vendor-specific manner using different structures and esoteric vocabularies. Tremendous manual effort is needed to annotate sensor nodes for each building or even to just segment these sensor names into meaningful chunks for intelligent operation of buildings. We propose here a *fully automated* self-supervised framework, Sensei, that can learn to segment sensor names without any human annotation. We employ a neural language model to capture the underlying structure in sensor names and then induce self-supervision based on information from the language model to build the segmentation model. Extensive experiments on five real-world buildings comprising thousands of sensors demonstrate the superiority of Sensei over baseline methods.

1 Introduction

Sensor name segmentation seeks to partition a sensor name string into semantic segments. It is an essential task to enable smart building applications (Weng and Agarwal, 2012) that critically depend upon understanding the context of sensory data. For example, to increase the flow of air to improve air quality, such an application will need to identify and locate the airflow controller for the specific area. Such a controller is identified as a control point associated with a specific actuator such as a variable air valve. This is typically done manually by searching appropriate strings in the name that refer to specific sensor or actuator type and/or its location. This information is encoded as a concatenation of *segments*. Correct segmentation of sensor names is a key first step. Note, we use sensor to refer to both sensors and actuators in a building application.

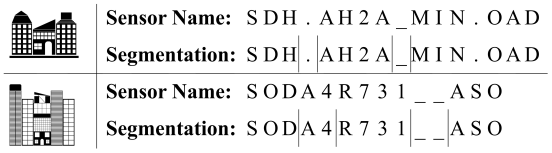

Figure 1: Sensor names adopt distinctive structures and vocabularies in different buildings, thus requiring manual effort to interpret.

Figure 1 shows examples of sensor names consisting of multiple segments, each encoding key context about the sensor (building name, location, sensor type, etc). Thus, the sensor name `SODA4R731__ASO` should be segmented as `SOD` (building name), `A4` (equipment id), `R731` (room id), and `ASO` (measurement type – area temperature setpoint). Note that the meanings of the same punctuation may vary; for example, '_' can be a delimiter or part of a segment.

Currently, sensor name segmentation requires domain knowledge and tedious manual effort due to its building-specific nature. Sensor names are created by building vendors, and as we see from Figure 1, in different buildings they usually adopt distinctive structures and vendor-specific esoteric vocabularies. Thus, constructing a sensor name segmentation model involves a building systems technician to comprehend these sensor names and then design rules to segment and annotate these names. There are no universal pre-defined parsing rules such as regular expressions for sensor names. This obscurity of sensor data remains a major obstacle to the wide adoption of smart building applications from both cost and efficiency perspectives (Bhattacharya et al., 2015a).

We need an automated solution for sensor name segmentation. Despite the recent progress in apply-

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 1017–1027

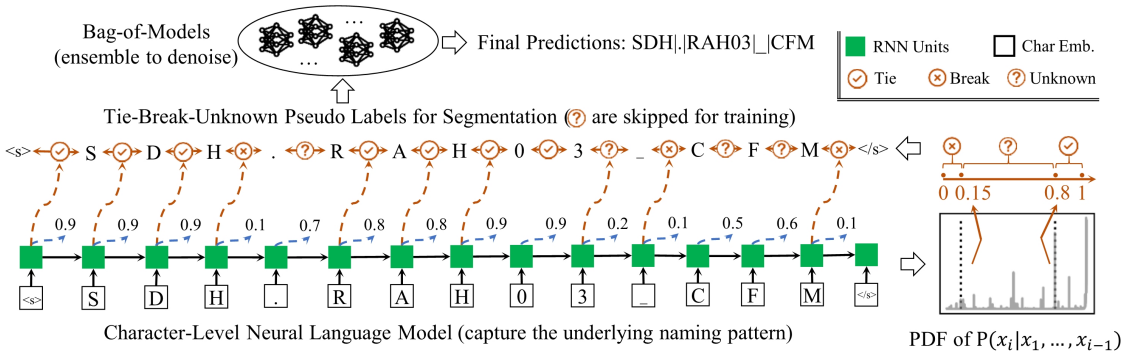

Figure 2: Overview of Sensei: We induce pseudo labels for segmentation using the transition probabilities from a character-level neural language model. Segmentation model training uses the hidden states from this model.

ing active learning (Schumann et al., 2014; Hong et al., 2015b; Balaji et al., 2015; Koh et al., 2018; Shi et al., 2019) and transfer learning (Hong et al., 2015a; Jiao et al., 2020) to sensor name interpretation, all these methods still require human annotation effort and thus they are not *fully* automated.

We propose here a novel self-supervised segmentation framework, Sensei, to segment sensor names into meaningful chunks *without any human effort*. This would enable understanding of important contextual information at scale and enable new regime of building applications. Figure 2 presents an overview.

Sensei builds upon the observation that sensor names are not randomly generated. Instead, installers of building management system would generally follow *some* underlying pattern for naming. For instance, in some buildings, the sensor name often starts with the building name, followed by the room id and type of measurement. Further, technicians would generally use similar phrases to express the same concept (e.g., "temperature" would be encoded as "T", "temp", or "ART"), at least within the same building. Based on this observation, we first employ a character-level neural language model (Karpathy et al., 2015) to capture the latent generative pattern in sensor names. This language model learns the probability of observing a character in the sensor name given all the preceding characters. Intuitively, the segment boundaries in a sensor name should highly correlate with this probability. Frequent transitions would have a higher probability than the infrequent ones, which might well imply the start of another segment. Therefore, we induce pseudo segmentation labels by setting a pair of thresholds on these transition probabilities, and then build a binary classifier to segment sensor names upon their contextualized representations produced by the language model. Since these pseudo labels may contain noise, we create an ensemble of independent classifiers, each trained on a uniformly random subset of the pseudo labels, in order to further improve the efficacy.

To the best of our knowledge, Sensei is the first framework for sensor name segmentation without human annotation. This paper makes the following contributions:

- We study an important problem of *fully automated* sensor name segmentation.
- We propose a novel self-supervised framework Sensei, which leverages a neural language model to capture the underlying naming patterns in sensor names and produces pseudo segmentation labels for training binary classifiers.
- We conduct extensive experiments on five real-world buildings comprising thousands of sensor names. Sensei on average achieves about 82% in F_1, roughly a 20-point improvement over the best compared unsupervised method.

Reproducibility. Our code and datasets are readily available on Github[1].

2 The Sensei Framework

Sensei consists of three steps:

- Train a neural language model (NLM) at the character level to capture the underlying naming patterns in sensor names;
- Generate `Tie-Break-Unknown` pseudo la-

[1] https://github.com/work4cs/sensei

bels using two thresholds, t_0 and t_1, decided by inspecting the distribution of transition probabilities (i.e., likelihood of observing the current character given the previous ones);
- Train a set of segmentation models based on the pseudo labels to mitigate the effect of noise in these labels.

We next elaborate on each step.

2.1 Language Model for Underlying Patterns

As sensor names are created by humans (e.g., a technician with knowledge about building particulars), they often follow a certain naming convention (e.g., start with the building name, then room id, and then type). The names also have an internal consistency, that is, within a building, segments of sensor names corresponding to the same kind of information (e.g., location or function) would use similar phrases; e.g., the concept of "room" would be encoded as "`RM`", "`R`", or similar variants. This prompts us to model the generative patterns in these names such that given the characters seen so far we can predict the next one. This coincides with the language modeling task in NLP.

Since the sensor name segmentation task works on characters, we adopt a popular character-level neural language model to capture the underlying sensor naming pattern, the classical Char-RNN (Karpathy et al., 2015) architecture, and use LSTM (Hochreiter and Schmidhuber, 1997) as the RNN model. Note that, our method is compatible with any character-level neural language models.

Given a character sequence of length N, $X = \langle x_1, x_2, \ldots, x_N \rangle$, the Char-RNN learns the probability of observing a character given all the previous characters, namely, $p(x_{i+1}|x_1, x_2, \ldots, x_i)$. During this process, we will obtain an embedding vector $\mathbf{x}_i$ for each character x_i, and a hidden state vector $\mathbf{h}_i$ after observing the characters from x_1 to x_i. A softmax layer is then applied to $\mathbf{h}_i$ to predict a distribution $\hat{\mathbf{p}}_\mathbf{i}$ over the entire vocabulary:

$$\hat{\mathbf{p}}_\mathbf{i}(c) = p(c|x_1, x_2, \ldots, x_i) = \frac{\exp\left(\mathbf{w}_c^\top \mathbf{h}_i\right)}{\sum_{c'} \exp\left(\mathbf{w}_{c'}^\top \mathbf{h}_i\right)},$$

where $\mathbf{w}_c$ is the linear transformation for character c. The cross-entropy between $\hat{\mathbf{p}}_\mathbf{i}$ and the one-hot encoding of x_{i+1} is used as the loss function for this character.

Given a building, we train the Char-RNN on all its sensor names. As each sensor name is independent of each other, we can have the same initial hidden state for each sensor name to ensure sensor names do not interfere with each other. Once the model converges, we apply it to all the sensor names to obtain the character transition probabilities, i.e., $\hat{\mathbf{p}}_\mathbf{i}(x_{i+1})$. The perplexity of the trained Char-RNN in our experiments is typically small (i.e., < 0.3 per batch with batch size 32). Therefore, we believe it captures the underlying naming pattern within the input building well.

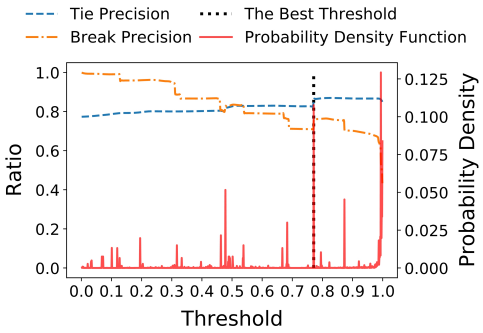

Figure 3: Plots of $\hat{\mathbf{p}}_\mathbf{i}(x_{i+1})$ histogram (grey bars) and Tie/Break precision curves for an example building. The "sweet spot", achieving a great balance between the tie- and break-precision scores, is highly aligned with the peak in the histogram.

2.2 Pseudo Labels from Transition Probabilities

Inspired by (Shang et al., 2018b), we use `Tie` and `Break` to decide the segmentation results. The transition between two adjacent characters (x_i, x_{i+1}) is labeled as (1) `Break` when we should segment after character x_i, or (2) `Tie` otherwise, denoting that the two successive characters belong to the same segment.

For a given character sequence $x_1, x_2, \ldots, x_N$, we hypothesize that the transition probability $\hat{\mathbf{p}}_\mathbf{i}(x_{i+1})$ obtained from Char-RNN is closely related to the `Tie/Break` relation between x_i and x_{x+1}. Intuitively, the Char-RNN model should produce a high likelihood for common transitions in sensor names, e.g., substrings for building name, room, and common sensor types. Therefore, when Char-RNN suggests a low transition probability, the transition is very likely to be a `Break`; otherwise, the possibility of a `Tie` becomes higher.

We empirically verify our hypothesis via data analysis of an example building as shown in Figure 3. We present the probability density from histogram of $\hat{\mathbf{p}}_\mathbf{i}(x_{i+1})$. In addition, based on the ground-truth segmentation results, we plot the Tie

and Break precision curves w.r.t. different thresholds. The *Tie Precision* refers to the ratio of Tie transitions among all the transitions above a certain threshold, while the *Break Precision* refers to the ratio of Break transitions among all the transitions below a certain threshold. One can observe that the"turning" points on the break precision curve are highly correlated to the peaks in the histogram. An uptick on the break precision curve indicates that there might be abundant same patterns in the bin of probabilities, which typically are more likely to be `Ties`. Therefore, the thresholds should be below the probability bins with high frequency to classify the steep upslope regions as `Tie`.

If one wants to set up a single threshold on $\hat{\mathbf{p}}_\mathbf{i}(x_{i+1})$ to classify all transitions into {`Tie`, `Break`} in an unsupervised manner, the highest peak in the "confidence" interval $[0.550, 0.950]$ on the distribution (e.g., 0.771 in Figure 3) would be a good choice to achieve a high F_1 score. We generalize this threshold selection criterion to the other buildings, and as we shall demonstrate in our experiments, such a selection strategy gives results close to grid search that uses ground-truth labels.

In addition to `Tie` and `Break`, we mark those uncertain transitions as `Unknown`. We need to decide on two thresholds, t_0 and t_1, and categorize the transitions according to three transition probability intervals, $[0, t_0]$, (t_0, t_1), and $[t_1, 1]$, denoting `Break`, `Unknown`, and `Tie`, respectively, as the *pseudo labels*. We wish these pseudo labels would be of high accuracy while having a sufficient amount of labels. Based on our observations, the above single threshold criterion satisfies t_1. Considering that `Breaks` are considerably fewer than `Ties`, we should decide on a `Break` more carefully. The highest peak in a narrowed high precision interval $[0.050, 0.150]$ would be appropriate (e.g., 0.101 in Figure 3).

2.3 Ensemble to De-noise Pseudo Labels

There could exist errors in these automatically induced pseudo labels, so we leverage the idea of ensemble learning to mitigate the impact of such errors on the final predictions (Breiman, 1996). Specifically, we independently sample a subset of pseudo labels to train K binary classifiers and then average their predictions. In the pseudo labels, the number of `Tie` transitions is usually much higher than that of `Break`. To balance the training data, we sample $\epsilon \cdot M$ `Tie` and `Break` labels, respectively, from all the pseudo labels, where M is the number of `Break` transitions and ϵ is a small coefficient between 0 to 1 for sampling a subset (e.g., $\epsilon = 0.1$). Such a sampling strategy makes the label errors less likely to affect every binary classifier, so the final prediction becomes more accurate.

All types of binary classifiers could be used to construct the ensemble. We adopt a Multi-Layer Perceptron (MLP) as the binary classifier. For the i-th transition, we retrieve the hidden state vector $\mathbf{h}_i$ yielded by the Char-RNN and feed it as input to the MLP. The final prediction is the average of predictions from the K classifiers. As the training data is sampled in a balanced way, we simply use 0.5 as the threshold to decide on `Tie` or `Break`.

3 Experiments

We empirically evaluate Sensei on datasets from real-world buildings and discuss our results as well as findings of particular interest.

3.1 Datasets and Pre-processing

We collected the sensor names from five office buildings contracted with four different building vendors at three different sites located in different geographic regions. We also collected the character-level ground-truth labels of these names from their building vendors. We adopted the `BIO` tagging scheme in generating labels, marking the beginning (`B`), inside (`I`), and outside (`O`) of each segment (e.g., for location or function). Table 1 summarizes details of each building.

Digits. The digits in sensor names indicate detailed and specific information such as room or equipment identifiers, so preserving the variety in numbers does not help our segmentation task. Conversely, it disturbs the transition probability distribution and thus confuses the model in predicting the next characters – the model would only need to learn and recognize the transitions from digit to digit, as opposed to the specific values (e.g., "`1`" to "`2`" or "`4`" to "`3`"). Therefore, we replace all numerical digits with the same digit "0".

Punctuation and Whitespace. There are symbols such as underscores and whitespace in sensor names that are inserted by technicians at the time of metadata construction. We leave them *as-is* for our model to learn their meanings because the meanings of these characters vary from case to case. This is, in fact, one of the major challenges in this

Table 1: Statistics of five buildings in our experiments. These builds are from three different campuses: Buildings **SDH** and **IBM** are from the first campus, **APM** and **EBU** from the second, and **SOD** from the third. Example sensor names are also listed for reference.

Building	#Sensors	#Segments	#Characters	Example Sensor Name
SDH	2,551	$2 \sim 5$	$7 \sim 31$	`SDH.AH1_RHC-4:CTL STPT`
IBM	1,366	$2 \sim 3$	$6 \sim 28$	`1F_FCU10_11_13_23_COLLAB`
APM	1,079	$1 \sim 7$	$4 \sim 34$	`AP&M-CRAC-2-MIG-009.COOLING ON-OFF`
EBU	1,074	$2 \sim 3$	$7 \sim 35$	`EBU3B.3RD FLR AVG CLG-PID1`
SOD	1,335	$2 \sim 4$	14	`SODC3P09DP_STA`

sensor name segmentation problem. For example, the sensor name "`SODH1_____L_L`" should be segmented as "`SOD|H1|_____|L_L`", with the three segments corresponding to its building name, equipment id, and measurement type, respectively. The underscores between "`H1`" and "`L_L`" are padded to make the sensor name fixed-length, while the underscore inside "`L_L`" connects two initial letters (i.e., for a Lead-Lag sensor, commonly existing in water pumps).

3.2 Evaluation Metrics

We evaluate the performance of all the considered methods by the F_1, precision, and recall scores. A segment is represented as a span with the starting and the ending character indices. A predicted segment is correct if and only if there exists an exactly same segment in the ground truth. Therefore, we define the precision and recall as follows:

$$\text{prec} = \frac{|\mathcal{S}_{GT}| \cap |\mathcal{S}_{Pred}|}{|\mathcal{S}_{Pred}|}, \text{rec} = \frac{|\mathcal{S}_{GT}| \cap |\mathcal{S}_{Pred}|}{|\mathcal{S}_{GT}|},$$

where $\mathcal{S}_{GT}$ is the set of ground-truth spans and $\mathcal{S}_{Pred}$ is the predicted set. The F_1 score is the harmonic mean of precision and recall. We report the averaged F_1 score of all sensor names, which is relatively unbiased (Opitz and Burst, 2019).

As we mentioned before, there will be some extra delimiters between segments. Therefore, during the evaluation, we ignore segments containing only delimiter(s) in both ground truth and predicted segments. When calculating the start and end indices for predicted segments, we also skip their prefix and suffix delimiters. The same process here applies to the evaluation of all methods.

3.3 Compared Methods

We compare Sensei with the following methods:

- **Delimiter**. There are punctuation (such as "-" and "_") and whitespace characters in sensor names, and they could indicate the boundaries between segments. Therefore, this method segments a sensor name by delimiters (i.e., non-alphanumeric characters). This method mainly serves as a sanity check.
- **NLTK TweetTokenizer**. NLTK (Bird et al., 2009) provides a tweet tokenizer to segment a string into tokens according to predefined regular expressions (regexes). We directly apply it to segment our sensor names.
- **CoreNLP**. We adopt the pre-trained tokenizer in the CoreNLP package[2] (Manning et al., 2014), which adopts the Universal Dependencies[3] version 2 (UD v2) standard for segmentation .
- **Stanza**. We also adopt Stanza[4] and use its built-in neural tokenizer (Qi et al., 2020) following UD v2. This method combines convolutional filters and bidirectional LSTM to realize tokenization and sentence segmentation as a tagging task (Qi et al., 2018).
- **N-gram-LM**. Character-level N-gram language model (N-gram-LM) can also provide character transition probabilities. To obtain an "upper bound" of the N-gram-LM's performance, we apply a grid search technique of Sensei-GS detailed in the following to find the optimal threshold on the test set. As most segments in the ground truth contain fewer than 5 characters, we tried N from 1 to 5 and reported on the best performance. However, this model does not offer any representations that we can use to train the ensemble classifiers for the "break/tie" decisions.
- **BayesSeg**. Topic segmentation divides a document into topic-coherent segments. An unsupervised Bayesian model, BayesSeg[5] (Eisenstein and Barzilay, 2008), is used to segment char-

[2] https://stanfordnlp.github.io/CoreNLP/

[3] Universal Dependencies is a framework of annotation guidelines by open community effort. https://universaldependencies.org/

[4] https://stanfordnlp.github.io/stanza/

[5] https://github.com/jacobeisenstein/bayes-seg

acters of sensor names as a topic segmentation task that decides the boundary between sentences. However, this method requires to manually specify the number of segments, which is a parameter we do not know without human input.

- **ToPMine**. ToPMine (El-Kishky et al., 2014) provides a method that groups frequent words into phrases in an unsupervised manner and incorporates these phrases into topic modeling. We adapt the model to work at the character level. That is, we regard each character of sensor names as a word in document and group characters into segments as group words into phrases.
- **SeNsER**. Besides the above unsupervised methods, we compare with a transfer learning model, SeNsER (Jiao et al., 2020), a supervised method that uses both source and target buildings' raw sensor names and source building's annotations to learn a sensor name tagger for use in the target building.

Note that, we do not use custom **regexes** to segment sensor names because they require tremendous manual effort to create in order to exhaustively cover all the possible substring patterns, which deviates from our self-supervised problem setting. Moreover, since different buildings follow different sensor naming conventions, manual effort is required from domain experts to create regexes on a per-building basis, which is a costly process.

We also compare with two ablations of our method:

- **Sensei-Forward (Sensei-FW)**. It leaves out the self-supervised ensemble learning. Specifically, we keep the Char-RNN to obtain the distribution of observing next characters, and then find the *single* threshold as stated in Section 2.2.
- **Sensei-Backward (Sensei-BW)**. This is similar to the forward counterpart. The only difference is that the Char-RNN takes as input the *reversed* sensor names. As we shall see in the results, this method does not add much value to our task due to the intrinsic irregularity of sensor names when examined backward.

We further examine a method using grid search based on ground truth for threshold tuning to verify the effectiveness of our threshold decision:

- **Sensei-GridSearch (Sensei-GS)**. Compared to Sensei-FW, this method finds the best threshold for deciding `Tie` using ground-truth labels, i.e., it searches through all the possible threshold values on the transition probability distribution and picks the one producing the best segmentation results. This method is only used to demonstrate that a single threshold chosen based on the transition distribution (as detailed in Section 2.2) gives results reasonably close to the best we can achieve for Sensei-FW using the ground truth.

3.4 Experimental Setup

We modify the Char-RNN library[6] and use Keras (Chollet et al., 2015) to implement our method. As our method is unsupervised, we do not employ the commonly used early-stopping scheme when training the Char-RNN. Instead, we train our models for 100 epochs and empirically find this to be sufficient. All the thresholds have three decimal places. We assign `Ties` as positives and `Breaks` as negatives. For binary classifier, any supervised learning algorithm (e.g., logistic regression, SVM, etc) would accommodate our need in this work. We choose a vanilla Multilayer Perceptron with 2 fully-connected layers, each with 64 cells. We set the number of binary classifiers in our ensemble, K, at 100. The subsampling rate for the ensemble, ϵ, is 10% and for each subsampling, we use pandas with the iteration index as seed. Training a Sensei model on a Colab GPU with 12GB RAM takes less than 40 minutes for each building. For the other compared methods, we tune at our best based on the recommended settings in their papers or repositories and report the best performance.

3.5 Main Results and Analysis

Experimental results for all the unsupervised methods are summarized in Table 2. Overall, Sensei significantly outperforms all the compared methods, attributed to its strategy of complementing the language model with a self-supervised ensemble classifier. Besides the variants of Sensei, the N-gram-LM baseline achieves the second-best performance among all the other unsupervised methods, with an average 62.13% in F_1 across all the buildings. This also illustrates the usefulness of a language model. By contrast, our Sensei achieves over 80% in F_1, which demonstrates a 20-point improvement over N-gram-LM.

When looking at the F_1 scores of baselines including Delimiter, ToPMine, BayesSeg, and the off-the-shelf tokenizers in NLTK, Stanza, and CoreNLP, they are not competitive; this highlights the need for a solution to our challenging problem.

[6] https://github.com/sherjilozair/char-rnn-tensorflow

Table 2: Performance of Sensei and compared methods on the five test buildings.

	SDH			IBM			APM			EBU			SOD			
Methods	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1	Avg F_1
Delimiter	33.21	47.18	38.47	52.61	65.87	57.80	3.10	4.44	3.56	32.00	46.60	37.73	46.54	23.95	31.51	33.81
NLTK	18.34	31.86	22.75	0.07	0.05	0.06	3.95	4.07	3.99	20.76	27.78	23.75	0.04	0.02	0.03	10.12
CoreNLP	17.09	13.46	14.75	0.04	0.02	0.03	39.31	30.88	34.20	15.86	10.58	12.69	0.11	0.06	0.07	12.35
Stanza	9.30	6.95	7.82	0.0	0.0	0.0	2.51	2.75	2.57	9.21	9.09	8.92	0.0	0.0	0.0	3.86
N-gram-LM	60.23	73.73	65.70	77.48	79.53	78.09	39.37	56.61	45.72	51.06	66.62	56.96	57.54	73.71	64.18	62.13
BayesSeg	1.74	2.17	1.92	19.72	28.54	23.25	9.72	10.18	9.88	15.05	25.16	18.82	45.07	34.16	38.84	18.54
ToPMine	16.83	31.42	21.76	27.83	38.86	31.86	14.39	30.63	19.46	2.11	4.55	2.85	15.38	26.17	19.27	19.04
Sensei-BW	10.93	11.00	9.18	0.0	0.0	0.0	0.98	3.86	1.53	1.04	4.66	1.69	19.22	11.33	13.77	5.23
Sensei-FW	61.17	74.45	66.56	39.97	53.40	44.84	38.58	55.58	44.81	47.94	64.65	53.78	58.38	74.18	64.87	54.97
Sensei-GS	61.17	74.45	66.56	79.84	80.43	79.76	38.58	55.58	44.81	47.94	64.65	53.78	58.38	74.18	64.87	61.91
Sensei	**87.00**	**83.64**	**84.95**	**84.81**	**90.80**	**86.84**	**70.23**	**77.98**	**73.21**	**78.10**	**85.77**	**80.39**	**85.81**	**87.53**	**86.43**	**82.36**

Table 3: Transfer learning performance among buildings by SeNsER (Jiao et al., 2020).

	SDH			IBM			APM			EBU			SOD		
Source Bldg	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1	Prec	Rec	F_1
SDH	99.87	99.79	99.83	87.11	76.28	81.34	91.20	83.72	87.30	70.10	69.30	69.73	29.75	39.23	33.84
IBM	78.79	85.72	82.11	99.80	99.90	99.85	76.07	59.23	66.60	63.70	66.12	64.89	29.45	39.71	33.82
APM	75.53	78.58	77.03	77.81	58.20	66.59	99.07	98.31	98.69	69.07	67.14	68.09	19.68	36.26	25.52
EBU	74.68	84.99	79.50	84.60	82.20	83.38	88.24	84.99	86.59	99.72	99.44	99.58	3.40	6.72	4.52
SOD	48.44	39.15	43.30	74.56	61.10	67.16	5.85	2.89	3.87	16.76	8.79	11.53	99.75	99.77	99.76

The performance of Delimiter confirms the fact that the semantics of these delimiters are mixed. If one recalls the examples in Table 1, vendors usually use delimiters in sensor names. Sometimes, these delimiters well indicate the segment boundaries. However, as we illustrated in the example sensor name "SOD|H1|_____|L_L", punctuation could be also used within the segment, and therefore simply segmenting at delimiters results in a considerable amount of false positives.

From Sensei-FW to Sensei, there is a significant boost, roughly 27 points in F_1 on average. Since the major difference between Sensei and Sensei-FW is our self-supervised ensemble learning module, we empirically verified its power.

Comparing Sensei-FW and Sensei-BW, we observe that the forward version performs dramatically better. Sensei-FW also performs better than Delimiter, ToPMine, and all the pre-trained tokenizers in all cases. By contrast, Sensei-BW takes the reversed sensor names as input but performs much worse than Sensei-FW. We notice that this is because there are not sufficient variations in the sensor string patterns when being looked at backward, compared to the forward case. For example, there are names like "SODA4R731__ASO" and "SODA1R516__VAV", and the Sensei-FW model can see various substrings (e.g., "ASO" and "VAV") following the common pattern "SODA0R000__". Variations as such provide enough information for the model to learn where to segment. However, when reversed, the above example becomes "OSA__000R0ADOS" and the *prefix* "OSA" sees no variations following, which makes it nearly impossible for Sensei-BW to figure out the right segmentation. Consequently, Sensei-FW better captures generative patterns while Sensei-BW achieves poor segmentation results.

Comparing Sensei-FW and Sensei-GS, one can observe that, in most cases (4 datasets out of 5), Sensei-FW finds the best single threshold found by Sensei-GS. Note that Sensei-GS utilizes the ground truth to exhaustively search among all the possible thresholds, while Sensei-FW decides the threshold based on the transition distribution without requiring any labels. This small difference in performance indicates that our data-driven threshold finding solution based on the distribution is reasonable and reliable.

From Table 3 we see that the performance of transfer learning varies drastically across buildings. This is because, as the method takes a transfer learning approach, its performance highly hinges on how similar the sensor names in the source building are to those in the target building. For example, **APM** and **EBU** are from the same vendor, and thus

Table 4: Performance of Sensei using different amounts of sensor names for training.

	SDH		IBM		APM		EBU		SOD	
Percentage (%)	#Sensors	F_1	#Sensors	F_1	#Sensors	F_1	#Sensors	F_1	#Sensors	F_1
25	637	72.67	341	75.95	269	39.99	268	33.50	333	57.86
50	1,275	92.28	683	71.84	539	48.99	537	47.77	667	85.62
75	1,913	86.38	1,024	85.04	809	57.61	805	70.45	1,001	85.31
100	2,551	84.95	1,366	86.84	1,079	73.21	1,074	80.39	1,335	86.43

SeNsER is relatively more effective on these two buildings. Moreover, since **APM** has more diverse patterns than **EBU** as shown in Table 4, it is reasonable that the transfer from **APM** to **EBU** results in a higher score than the opposite. Additionally, **SOD** are so different from all the other buildings that transfers to **SOD** produce poor results. The noticeable high diagonal scores, whose source and target buildings are the same (i.e., learning within a building), provide an upper-bound reference. We shall also note that, referring to the last row in Table 2, Sensei outperforms SeNsER on four buildings except for **APM**, as Sensei is focused on more effectively capturing patterns inside a building.

3.6 Effect of Number of Sensors

As Sensei framework is fully automated in a self-supervised manner, its performance is solely affected by the amount and variety of sensor names available for learning the segmentation classifier. As shown in Table 4, Sensei generally gets better performance with more sensor names available with an exception of Building **SDH**. We hypothesize that the performance relates more closely to the variety of sensor name patterns in the dataset rather than the number of sensor names.

3.7 Case Studies and Discussions

We next showcase some examples that Sensei correctly segments, in order to illustrate its capability.

"Flukes" for False Positives. In Building **IBM**, some of the `Breaks` are recognized as `Ties` by Sensei-FW and Sensei-GS. For example,

`0F|_|SRVC|_|D0D0D0D00`,
`GF|_|SRVC|_|QR000_000`,

are mistakenly segmented as

`0F_SRVC|_|D0D0D0D00`,
`GF_SRVC|_|QR000_000`.

By contrast, Sensei avoids the mistakes by learning the pattern from many other sensor names. The following case is a great example.

`GF|_|LGHT|_|COFFEEDOCK`.
`GF|_|FRONTAISLE|_|LHS`,
`0F|_|FCU_KWH`.

There are only 89 occurrences of "`_|LGHT|`" compared to 177 of "`_|SRVC|`". Thus, with a lower transition probability, it can be recognized as a `Break` before "`_|LGHT|`". Many similar cases can teach Sensei that `Break` is more likely in this pattern, facilitating its performance.

"Flukes" for False Negatives. Building **SOD** contains many cases as follows:

`SOD|A0|R000|__|ASO`,
`SOD|A0|R000|__|AGN`.

Sensei-FW, and even Sensei-GS which employs the ground truth, are not able to segment these names correctly; they instead segment them as

`SOD|A0|R000|__|A|SO`,
`SOD|A0|R000|__|A|GN`,

because of the same prefix "`SODA0R000__A`".

By contrast, Sensei is able to correctly segment them owing to the self-supervised ensemble learning, which is more robust to noise in pseudo labels.

Discussion. We notice that even though Sensei on average achieves about 80% in F_1, it still has limitations. Sensei is sensitive to the variation of patterns in datasets—the patterns cannot be too varied or too monotonous.

4 Related Work

There are three lines of prior and related work, namely, sensor metadata mapping, language model, and phrase mining.

Sensor Metadata Tagging. Sensor Metadata Tagging refers to the process of parsing and annotating the sensor metadata (or sensor name) for understanding a sensor's key context, including the measurement type (Balaji et al., 2015; Hong et al., 2015b), location (Bhattacharya et al., 2015b), relationships with others (Koh et al., 2018), and many more (Schumann et al., 2014). The majority body

of work exploits an active learning-based procedure (Settles, 2009), where it iteratively selects an "informative" and "representative" metadata example for a domain expert to label, in order to learn a model to annotate the metadata. Complementary to the use of textual metadata, there are also efforts exploring the use of time-series data for inferring the sensor context (Koc et al., 2014; Pritoni et al., 2015; Hong et al., 2015a). While they can significantly reduce the required manual labeling, they still rely on the availability of at least one human annotator to segment, parse, and provide labels.

By contrast, the method proposed in this work is fully automated, i.e., completely removing humans from the process, and we demonstrate its use in an essential first step—segmenting a sensor name string into meaningful substrings.

Language Model and Tokenization. Language models originate from the areas of natural language processing and information retrieval (Schütze et al., 2008). They aim at modeling the likelihood of observing one token given all the tokens before it, capturing the underlying language patterns. Recent advances in deep learning have pushed the language modeling from traditional n-gram models to neural language models (Kiros et al., 2014; Karpathy et al., 2015; Kim et al., 2016; Peters et al., 2018; Devlin et al., 2018), achieving significantly better performance using recurrent neural networks.

Analogizing sensor names to human languages, we employ neural language models to capture the underlying naming pattern. As we seek to segment a sensor name string into substrings, we choose the classic Char-RNN model (Karpathy et al., 2015). In general, any character-level language models are applicable in our method.

One can also view our problem as tokenization of sensor names. We thus compare with multiple existing tokenizers provided in NLTK Twitter, Standford CoreNLP (Manning et al., 2014), and Stanza (Qi et al., 2020). As we demonstrate in the evaluation, our method significantly outperforms these methods in segmenting sensor names.

Phrase Mining. Treating characters as words, our problem can be viewed as an unsupervised phrase mining problem with phrasal segmentation as output. Existing methods mainly leverage statistical signals based on term frequency in the corpus (Deane, 2005; Parameswaran et al., 2010; Danilevsky et al., 2014; El-Kishky et al., 2014). Among all these methods, ToPMine (El-Kishky et al., 2014) is arguably the most effective one. Our method Sensei significantly outperforms ToPMine in our empirical evaluation. There exist weakly/distantly supervised phrase mining methods (Liu et al., 2015; Shang et al., 2018a); however, such supervision signals (e.g., Wikipedia) are difficult to obtain for building sensors.

5 Conclusions and Future Work

Smart buildings critically rely upon contextual information associated with its sensors and actuators to appropriately sense and operate building subsystems. Such contextual information appears in the form of sensor metadata that is part of the installation of building management system. In this paper, we have presented Sensei, a system for automated segmentation of metadata information. Sensei is a fully automated method without requiring human labels that employs a character-level neural language model to capture the underlying generative patterns in building sensor names. Based on the probability distribution of character transitions (i.e., likelihood of observing the current character give the previous ones), it decides on two thresholds for sifting out examples for which it is confident to be `Tie` or `Break`. Considering these pseudo-labeled examples as supervision, Sensei constructs an ensemble of binary classifiers to segment sensor names with the information provided by the language model. We conducted experiments on the sensor names from five real-world buildings, and Sensei on average achieves $F_1 > 80\%$ in segmenting sensor names, a roughly 20-point improvement over the best compared unsupervised method. Our ongoing work addresses pre-training methods for the language model to improve Sensei's performance and its use in standard NLP tasks.

Acknowledgement

We thank reviewers for the anonymous comments and suggestions to improve this work. This work was supported in part by National Science Foundation 1940291 and 2040727. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and should not be interpreted as necessarily representing the views, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright annotation hereon.

References

Bharathan Balaji, Chetan Verma, Balakrishnan Narayanaswamy, and Yuvraj Agarwal. 2015. Zodiac: Organizing large deployment of sensors to create reusable applications for buildings. In *BuildSys*, pages 13–22. ACM.

Arka Bhattacharya, Joern Ploennigs, and David Culler. 2015a. Short paper: Analyzing metadata schemas for buildings: The good, the bad, and the ugly. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 33–34. ACM.

Arka A Bhattacharya, Dezhi Hong, David Culler, Jorge Ortiz, Kamin Whitehouse, and Eugene Wu. 2015b. Automated metadata construction to support portable building applications. In *BuildSys*, pages 3–12. ACM.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24(2):123–140.

François Chollet et al. 2015. Keras. https://keras.io.

Marina Danilevsky, Chi Wang, Nihit Desai, Xiang Ren, Jingyi Guo, and Jiawei Han. 2014. Automatic construction and ranking of topical keyphrases on collections of short documents. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 398–406. SIAM.

Paul Deane. 2005. A nonparametric method for extraction of candidate phrasal terms. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 605–613.

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*.

Jacob Eisenstein and Regina Barzilay. 2008. Bayesian unsupervised topic segmentation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 334–343.

Ahmed El-Kishky, Yanglei Song, Chi Wang, Clare R Voss, and Jiawei Han. 2014. Scalable topical phrase mining from text corpora. *Proceedings of the VLDB Endowment*, 8(3):305–316.

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

Dezhi Hong, Hongning Wang, Jorge Ortiz, and Kamin Whitehouse. 2015a. The building adapter: Towards quickly applying building analytics at scale. In *BuildSys*.

Dezhi Hong, Hongning Wang, and Kamin Whitehouse. 2015b. Clustering-based active learning on sensor type classification in buildings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 363–372. ACM.

Yang Jiao, Jiacheng Li, Jiaman Wu, Dezhi Hong, Rajesh Gupta, and Jingbo Shang. 2020. SeNsER: Learning cross-building sensor metadata tagger. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 950–960.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Ryan Kiros, Ruslan Salakhutdinov, and Rich Zemel. 2014. Multimodal neural language models. In *International Conference on Machine Learning*, pages 595–603.

Merthan Koc, Burcu Akinci, and Mario Bergés. 2014. Comparison of linear correlation and a statistical dependency measure for inferring spatial relation of temperature sensors in buildings. In *BuildSys*, pages 152–155. ACM.

Jason Koh, Bharathan Balaji, Dhiman Sengupta, Julian McAuley, Rajesh Gupta, and Yuvraj Agarwal. 2018. Scrabble: transferrable semi-automated semantic metadata normalization using intermediate representation. In *Proceedings of the 5th Conference on Systems for Built Environments*, pages 11–20. ACM.

Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, and Jiawei Han. 2015. Mining quality phrases from massive text corpora. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1729–1744.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Juri Opitz and Sebastian Burst. 2019. Macro f1 and macro f1. *arXiv preprint arXiv:1911.03347*.

Aditya Parameswaran, Hector Garcia-Molina, and Anand Rajaraman. 2010. Towards the web of concepts: Extracting concepts from large datasets. *Proceedings of the VLDB Endowment*, 3(1-2):566–577.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Marco Pritoni, Arka A Bhattacharya, David Culler, and Mark Modera. 2015. Short paper: A method for discovering functional relationships between air handling units and variable-air-volume boxes from sensor data. In *BuildSys*, pages 133–136. ACM.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. *arXiv preprint arXiv:2003.07082*.

Anika Schumann, Joern Ploennigs, and Bernard Gorman. 2014. Towards automating the deployment of energy saving approaches in buildings. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 164–167. ACM.

Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference*, volume 4.

Burr Settles. 2009. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Jingbo Shang, Jialu Liu, Meng Jiang, Xiang Ren, Clare R Voss, and Jiawei Han. 2018a. Automated phrase mining from massive text corpora. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1825–1837.

Jingbo Shang, Liyuan Liu, Xiaotao Gu, Xiang Ren, Teng Ren, and Jiawei Han. 2018b. Learning named entity tagger using domain-specific dictionary. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2054–2064.

Zixiao Shi, Guy R Newsham, Long Chen, and H Burak Gunay. 2019. Evaluation of clustering and time series features for point type inference in smart building retrofit. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 111–120.

Thomas Weng and Yuvraj Agarwal. 2012. From buildings to smart buildings—sensing and actuation to improve energy efficiency. *IEEE Design & Test of Computers*, 29(4):36–44.