\Orchestrating a brighter world

DeepJoin: Joinable Table Discovery with Pre-trained Language Models

Yuyang Dong, Chuan xiao Takuma Nozawa, Masafumi Enomoto, Masafumi Oyamada

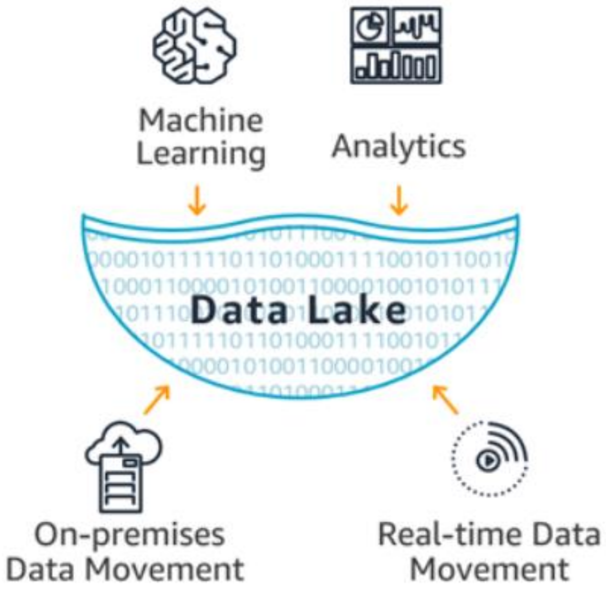
大阪大学
OSAKA UNIVERSITY

Background: what is a data lake?

- Data lake is a data repository that stores a large of data.
- In this work, we focus on how to efficient discovery tabular data (e.g., csv tables) from large table sets like data lake.

Race	Population	Median Age
White	234,370,202	42.0
Black	40,610,815	32.7
American Indian/ Alaska Native	2,632,102	31.7
Hawaiian/ Guamanian/Samoan	570,116	29.7

Population



https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/

Background: what is table join?

◆ Join is an essential operation that connect two or more tables

join

Race	Income
White	65,902
Black	41,511
Mainland Indigenous	44,772
Pacific Islander	61,911

Median household income

Race	Population	Median Age
White	234,370,202	42.0
Black	40,610,815	32.7
American Indian/ Alaska Native	2,632,102	31.7
Hawaiian/ Guamanian/Samoan	570,116	29.7

Population

equi-join

Race	income	Population	Median Age
White	65,902	234,370,202	42.0
Black	41,511	40,610,815	32.7

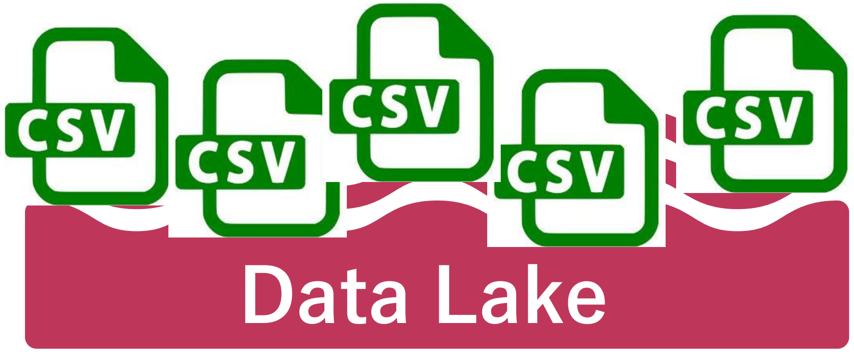
Joinable table discovery Problem

- **Given a query table, find joinable tables from data lakes**

query column

Name	Col_1	Col_2
…		
…		
…		

query table



joinable tables

- **Applications:**

Data recommendation

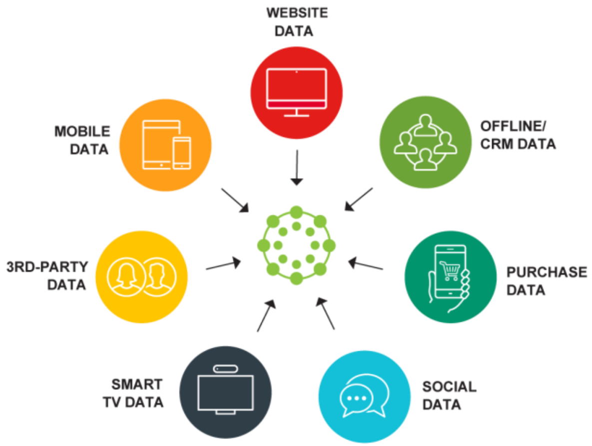
Name	Att1	Att2
…		
…		
…		

join

Name	Att3	Att4
…		
…		
…		

enrich

Data enrichment



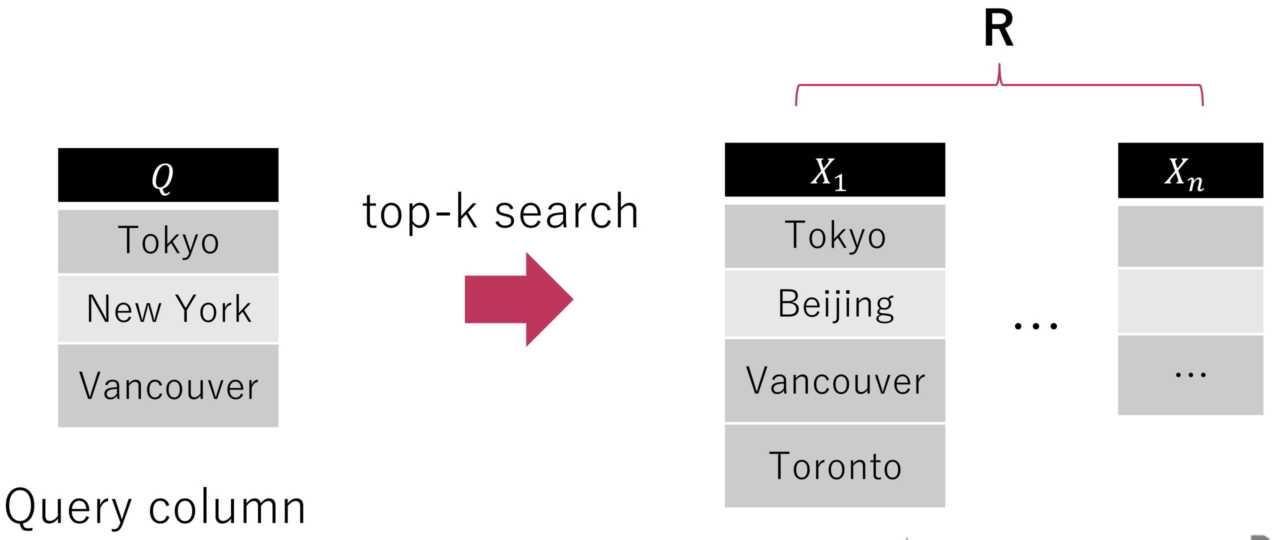
Data management

Problem definition

- Given a query column $\boldsymbol{Q}$, a collection of columns $\boldsymbol{R}$
- Find top-k columns with the highest joinability $\mathbf{J}(.)$

- Joinability between $\boldsymbol{Q}$ and a target column $\boldsymbol{X}$: $\mathbf{J}(\mathbf{Q}, \mathbf{X}) = |\boldsymbol{Q_M}| \ / \ |\boldsymbol{Q}|$
- $\boldsymbol{Q_M}$ is the matching records between $\boldsymbol{Q}$ and $\boldsymbol{X}$
- For example, find tables with equi-join: $\boldsymbol{Q_M} = \boldsymbol{Q} \cap \boldsymbol{X}$

$Q \cap X_1$ = {“Tokyo, “Vancouver”}
$J(Q, X_1) = 2/3$



Research motivation: general and efficient

- General: Need support any kind of joinability
  - Idea : A **learning base approach** can adjust different joinability with different training data

	Join-type	Approach	Problem
LSH-Ensemble VLDB' 16	Equi	Rule-base minhash-LSH	Threshold Search
JOSIE SIGMOD' 19	Equi	Rule-base inverted-index	Top-k search
PEXESO ICDE' 21	Semantic	Rule-base tree-base-index + inverted index	Threshold Search
NextiaJD, EDBT' 21	Any	Learning-base Random forest	Classification
DLN, VLDB' 21	Any	Learning-base Random forest	Classification
Deep-join	Any	Learning-base Pretrain language model + Ann index	Top-k Search

Research motivation: general and efficient

- Efficient: consider both accuracy and speed

- Accuracy
  - Need a good model to predict joinability correctly
    - Idea 1: Using Pretrained Language Model (**PLM**)

- Speed problem

Research motivation: general and efficient

- Efficient: consider both accuracy and speed

- Accuracy
  - Need a good model to predict joinability
    - Idea 1: Using Pretrained Language Model (**PLM**)

- Speed problem
  - If predict with pairwise column -> O(n), too slow and too cost
    - Idea 2:
      - Using Embedding based retrieval with **PLM encoder**
      - Efficient search of top-k embedding vectors with **ANN index** -> O(logn)

Research motivation: general and efficient

- Efficient: consider both accuracy and speed

- Accuracy
  - Need a good model to predict joinability
    - Idea 1: Using Pretrained Language Model (**PLM**)
  - Need a good embedding for column joinability
    - Idea 3: PLM encoder + **Metric learning**

- Speed problem
  - If predict with pairwise column -> O(n), too slow
    - Idea 2:
      - Using Embedding based retrieval with **PLM encoder**
      - Search top-k with **ANN index** -> O(logn)

Research motivation: general and efficient

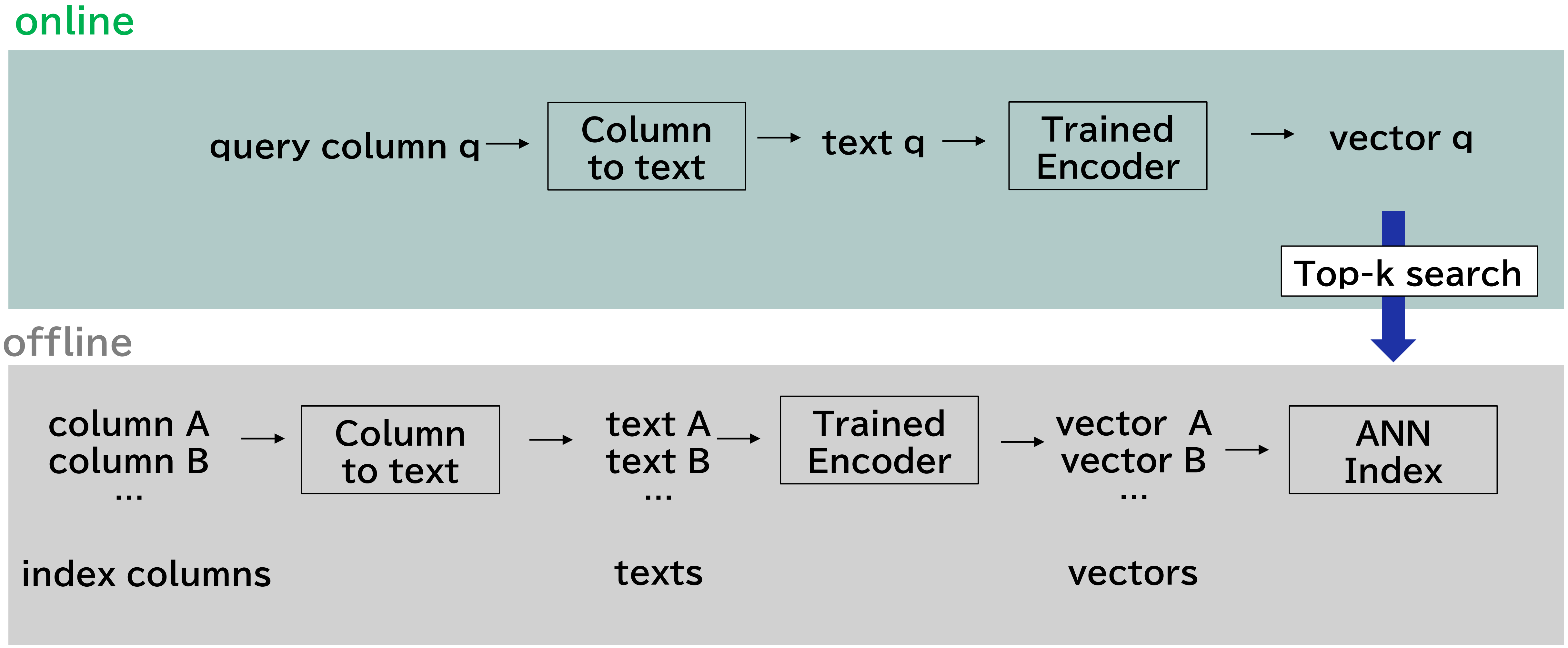
- Efficient: consider both accuracy and speed

- Accuracy
  - Need a good model to predict joinability
    - Idea 1: Using Pretrained Language Model (**PLM**)
  - Need a good embedding for column joinability
    - Idea 3: PLM encoder + **Metric learning**

- Speed problem
  - If predict with pairwise column -> O(n), too slow
    - Idea 2:
      - Using Embedding based retrieval with **PLM encoder**
      - Search top-k with **ANN index** -> O(logn)

→ Deep-join

Overview


online
query column q
Column to text
text q
Trained Encoder
vector q
Top-k search
offline
column A
column B
…
Column to text
text A
text B
…
Trained Encoder
vector A
vector B
…
ANN Index
index columns
texts
vectors

Column to text

◆ Column to text for LM

Pattern name	Template
col	$col_records$
colname-col	col_name : $col_records$.
colname-col-intotal	col_name : $col_records$. In total n unique records.
colname-col-context	col_name : $col_records$. $tab_context$
title-colname-col	tab_title. col_name : $col_records$.
title-colname-col-intotal	tab_title. col_name : $col_records$. In total n unique records.

tab_context

The Population Census shows that Japan had 55.70 million private households. The Population Census shows that Japan had 55.70 Of that total, 54.2 percent were nuclear-family households, and 38.1 percent were one-person households.

col_name

col_records

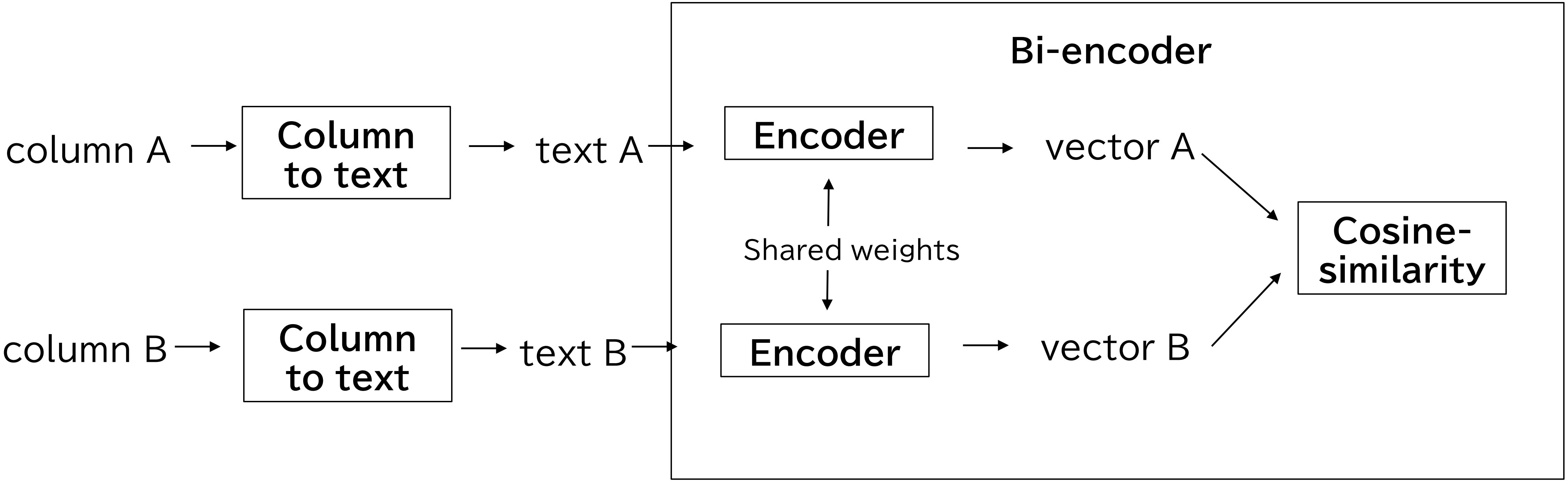
City	Population	Area
Tokyo		
Tsukuba		
Nagoya		
Kawasaki		

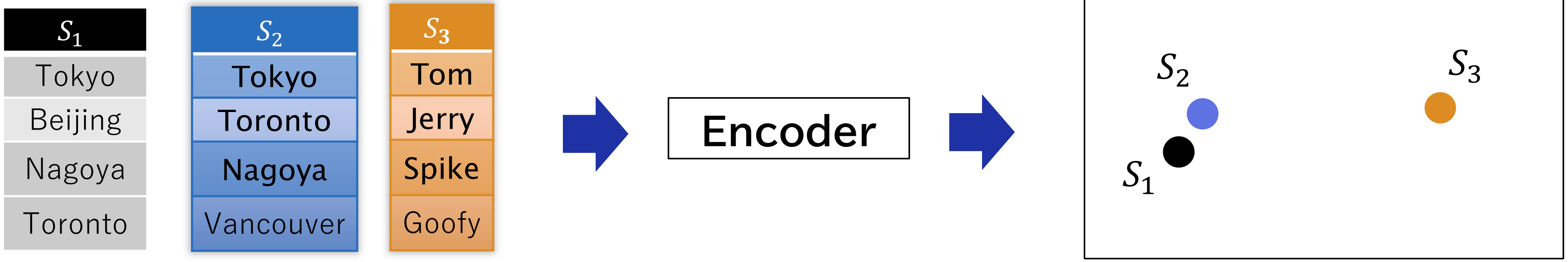
Statistics of Japanese cities

tab_title

Text: <u>“City</u> of <u>statistics of Japanese cities</u> contains <u>4</u> values: <u>Tokyo, Tsukuba, Nagoya, Kawasaki”</u>

Bi-encoder architecture





Training

- **Loss function**
  - Multiple negative ranking loss
  - Minimize the approximated mean negative log probability

$$
\begin{aligned}
L(\mathrm{X},\mathrm{Y}) &= -\frac{1}{N}\sum_{i=1}^{N}\log P_{\text{approx}}(Y_i \mid X_i) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\left[S(X_i,Y_i), -\log\sum_{j=1}^{N}\exp\left(S(X_i,Y_j)\right)\right].
\end{aligned}
$$

- **Negative sample**
  - In-batch random negatives (In exp. better than add some hard negative)
- **Data augmentation**
  - Random shuffling the cells in column

Experiment setting

- **Dataset**
  - WDC webtable
  - Wikitable
  - Sample 1M columns and 20K training positive pairs (equi, semantic)
- **Compare methods**
    - Minhash-lsh (VLDB16)
    - JOSIE (SIGMOD19)
    - Fasttext, BERT, MPNET emb
    - TaBERT (ACL20)
    - TURL (VLDB20)
- **Deep-join implement**
  - Bi-encoder: sentence-BERT
  - Encoder: BERT, MPNet
  - ANN index: FAISS IVFPQ-HNSW

Experiment: Accuracy

- Deep-join is better than compared methods with up to +15% in pre@k and +16% in NDCG@k

Table 3: Accuracy of equi-joins.

	Precision@k					NDCG@k				
Methods	k = 10	20	30	40	50	k = 10	20	30	40	50
Webtable										
LSH Ensemble	0.634	0.647	0.656	0.676	0.688	0.715	0.714	0.701	0.702	0.698
fastText	0.680	0.726	0.752	0.754	0.773	0.731	0.721	0.743	0.748	0.764
BERT	0.652	0.695	0.712	0.722	0.729	0.698	0.713	0.708	0.707	0.708
MPNet	0.610	0.629	0.644	0.649	0.654	0.674	0.677	0.678	0.680	0.677
TaBERT	0.622	0.637	0.645	0.656	0.671	0.694	0.685	0.690	0.693	0.691
TURL	0.653	0.669	0.689	0.711	0.721	0.688	0.706	0.716	0.727	0.732
MLP	0.683	0.719	0.755	0.758	0.778	0.737	0.735	0.748	0.755	0.769
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.702	0.741	0.775	0.793	0.805	0.744	0.752	0.758	0.761	0.788
DeepJoin$_{\text{MPNet}}$ (ours)	**0.732**	**0.775**	**0.791**	**0.812**	**0.832**	**0.768**	**0.786**	**0.799**	**0.803**	**0.822**
Wikitable										
LSH Ensemble	0.480	0.450	0.466	0.470	0.474	0.714	0.688	0.681	0.674	0.672
fastText	0.574	0.551	0.581	0.605	0.621	0.799	0.794	0.791	0.793	0.791
BERT	0.436	0.460	0.497	0.520	0.541	0.719	0.721	0.731	0.736	0.740
MPNet	0.442	0.464	0.504	0.524	0.543	0.711	0.721	0.729	0.735	0.736
TaBERT	0.431	0.445	0.488	0.520	0.539	0.701	0.708	0.732	0.725	0.737
TURL	0.504	0.525	0.529	0.545	0.578	0.707	0.711	0.745	0.766	0.778
MLP	0.578	0.576	0.585	0.610	0.619	0.801	0.802	0.800	0.804	0.802
DeepJoin$_{\text{DistilBERT}}$ (ours)	0.588	0.593	0.612	0.635	0.807	0.813	0.822	0.825	0.823	0.827
DeepJoin$_{\text{MPNet}}$ (ours)	**0.614**	**0.622**	**0.641**	**0.666**	**0.678**	**0.821**	**0.824**	**0.830**	**0.833**	**0.833**

Experiment: Speed

- Embedding based retrieval with ANN index is over 10x faster than traditional minhash-LSH and inverted-index
- Deep-join needs encode query online
  - CPU environment
    - slower than fasttext
  - GPU environment
    - Similar level of speed to fasttext

Table 14: Processing time per query, varying k.

Methods	query encoding (ms)	total (ms)				
		k = 10	20	30	40	50
Webtable, equi-joins						
LSH Ensemble [61]	-	496	506	590	595	508
JOSIE [59]	-	535	556	578	580	506
fastText	9	10.3	10.5	10.2	10.8	11.1
DeepJoin (CPU)	66	67.1	67.1	67.1	67.2	68.1
DeepJoin (GPU)	7	8.4	8.1	8.2	8.1	8.0
Webtable, semantic joins						
PEXESO	-	2345	2444	2356	2754	2566
DeepJoin (CPU)	74	75.6	76.8	76.1	75.8	76.1
DeepJoin (GPU)	7	8.1	8.3	8.0	8.2	8.4
Wikitable, equi-joins						
LSH Ensemble [61]	-	652	720	715	678	736
JOSIE [59]	-	647	667	708	697	788
fastText	6	7.4	7.2	7.8	7.3	7.7
DeepJoin (CPU)	76	77.1	78.1	77.4	77.5	77.6
DeepJoin (GPU)	5	6.3	7.0	6.6	6.7	6.4
Wikitable, semantic joins						
PEXESO	-	2655	2776	2557	2743	2789
DeepJoin (CPU)	86	87.4	87.3	87.1	87.2	87.7
DeepJoin (GPU)	9	10.5	11	10.2	10.7	10.4

Thanks! Question?

\Orchestrating a brighter world

NEC