Subword-based Compact Reconstruction of Word Embeddings

Shota Sasaki, Jun Suzuki, Kentaro Inui

RIKEN AIP, Tohoku University

Quick overview

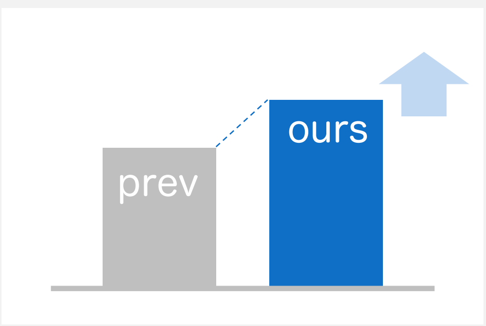
Proposal: novel word embeddings

OPEN-vocabulary

COMPACT
model size

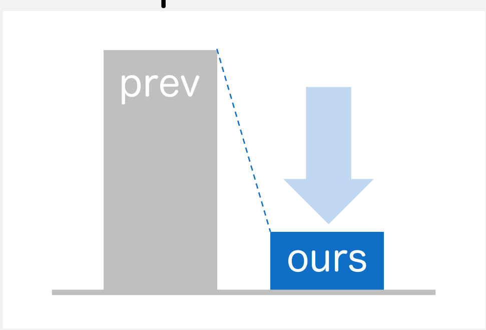
Performance:

✔ Better score



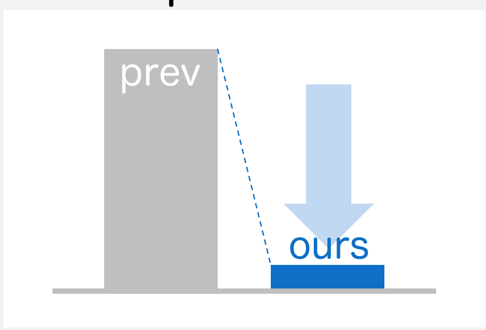
WordSim (OOV)

✔ 1/4 model size
✔ Comparable score



Model Compression Test

✔ 1/10 model size
✔ Comparable score



Downstream Tasks

Outline

- Quick Overview
- Background
  - Word embeddings
  - Related work
  - Purpose
- Proposed Method
  - Key technique
    - Subword-to-memory mapping function
    - Subword mixing function
- Experiments
  - Word similarity with OOV words
  - Model compression test
  - Downstream tasks (NER, TE)

Background: Pretrained Word Embeddings

✓ **Highly beneficial, fundamental language resources**

- e.g., **GloVe.840B** embeddings [Pennington, 2014]
  - Training data: **Common Crawl Corpus** (840B tokens)
  - Available online

✗ Inapplicability to out-of-vocabulary (OOV) words

- Infrequent words (often cut off due to memory requirements)
- Novel words

Background: Pretrained Word Embeddings

✓ **Highly beneficial, fundamental language resources**

- e.g., **GloVe.840B** embeddings [Pennington, 2014]
  - Training data: **Common Crawl Corpus** (840B tokens)
  - Available online

✗ **Inapplicability to out-of-vocabulary (OOV) words**

- Novel words
- Infrequent words (often cut off due to memory requirements)

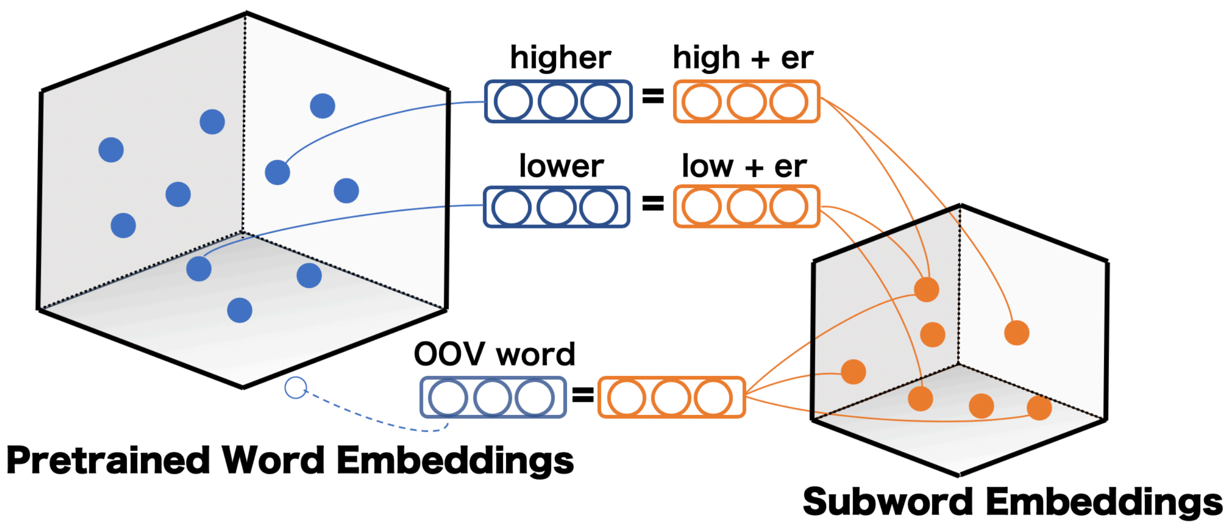
Related Work:
Bag of Subwords (BoS) [Zhao et al., EMNLP-2018]

- **Similar motivation**

  Reconstruct pretrained word embeddings
  to support out-of-vocabulary (OOV) words

- **Basic Idea**

  Compute embeddings of OOV words by summing up
  subword embeddings obtained through the reconstruction



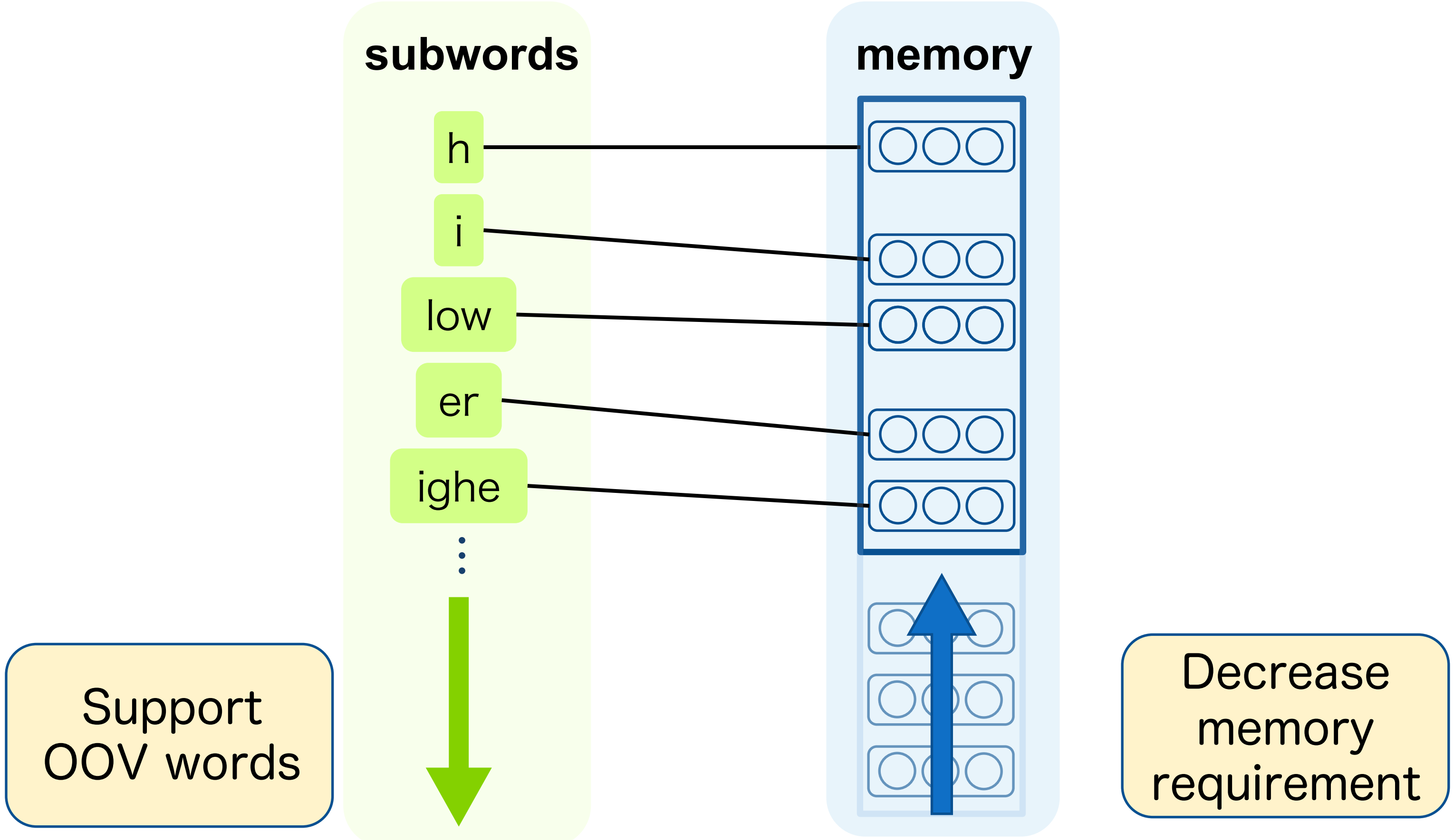
The Problem with Subwords: There are too many

Naïve approach significantly increases memory requirements

Setting	# of vectors (aka. # of vocab.)	Memory
Pre-trained word embeddings (fastText.600B)	2.0 M	2.2 GB
char N-gram (N=1, 2, ..., 6) subword embeddings	6.3 M	7.2 GB

Mem. (GB) = # of vectors × # of dimensions × 4bytes (float) / 1024^3

Purpose



Aim to develop a method that simultaneously satisfies
①less memory requirement ②applicability of OOV

Outline

- Quick Overview
- Background
  - Word embeddings
  - Related work
  - Purpose
- Proposed Method
  - Key technique
    - Subword-to-memory mapping function
    - Subword mixing function
- Experiments
  - Word similarity with OOV words
  - Model compression test
  - Downstream tasks (NER, TE)

Outline

- Quick Overview
- Background
  - Word embeddings
  - Related work
  - Purpose
- Proposed Method
  - Key technique
    - Subword-to-memory mapping function
    - Subword mixing function
- Experiments
  - Word similarity with OOV words
  - Model compression test
  - Downstream tasks (NER, TE)

Key Ideas

- Subword-to-memory mapping function
  1. Discarding infrequent subwords
  2. Memory sharing
  3. Combination of 1. and 2.

- Subword mixing function
  - Self-attention mechanism

Key Ideas

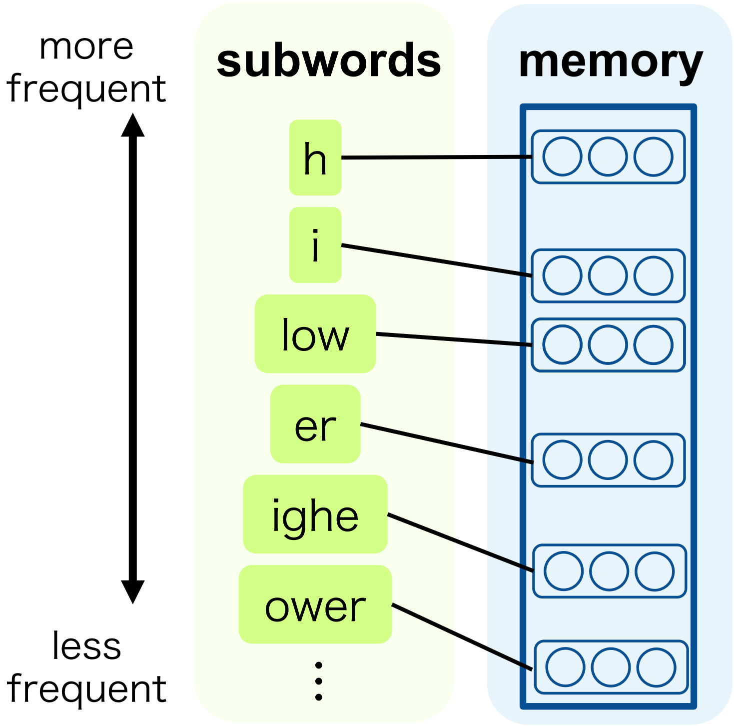
- Subword-to-memory mapping function
  1. Discarding infrequent subwords
  2. Memory sharing
  3. Combination of 1. and 2.

- Subword mixing function
  - Self-attention mechanism

Subword-to-Memory Mapping

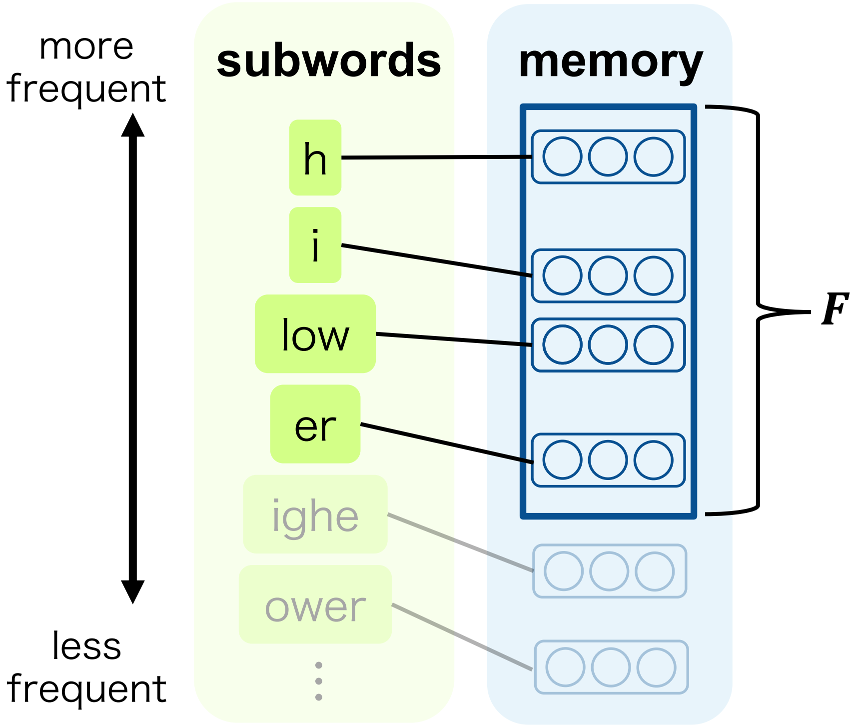
1. **Discarding infrequent subwords**

- Use top-F frequent subwords instead of all possible subwords
- Model size $= F \times$ # of dimensions



1. **Discarding infrequent subwords**

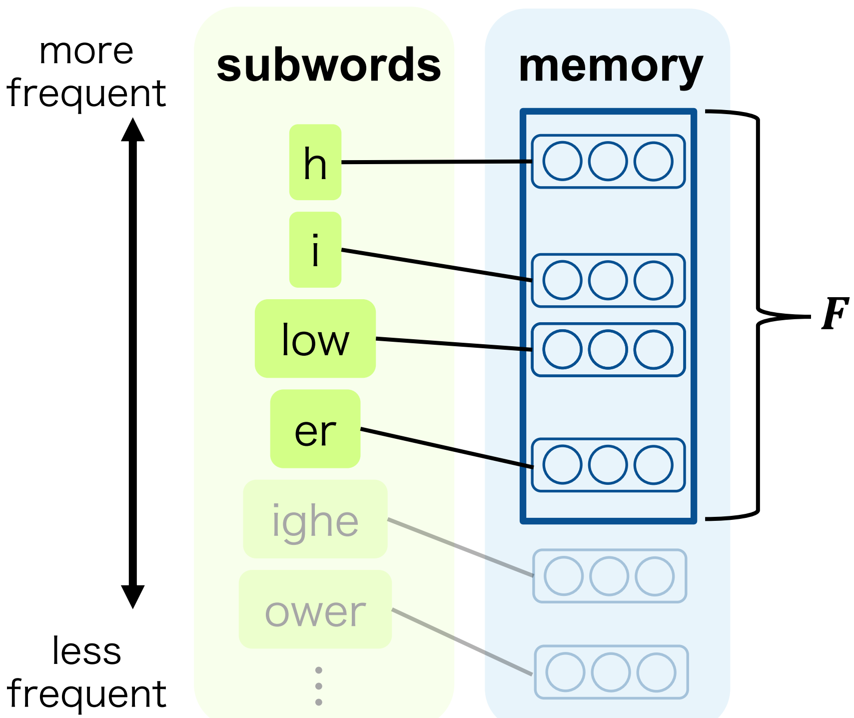
- Use top-F frequent subwords instead of all possible subwords
- Model size = $F \times$ # of dimensions



Subword-to-Memory Mapping

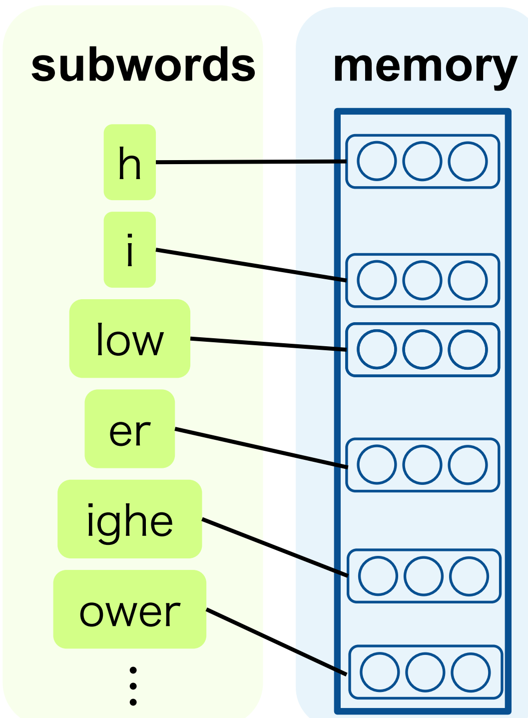
1. Discarding infrequent subwords

- Use top-F frequent subwords instead of all possible subwords
- Model size $= F \times$ # of dimensions



2. Memory sharing [Bojanowski, 2017]

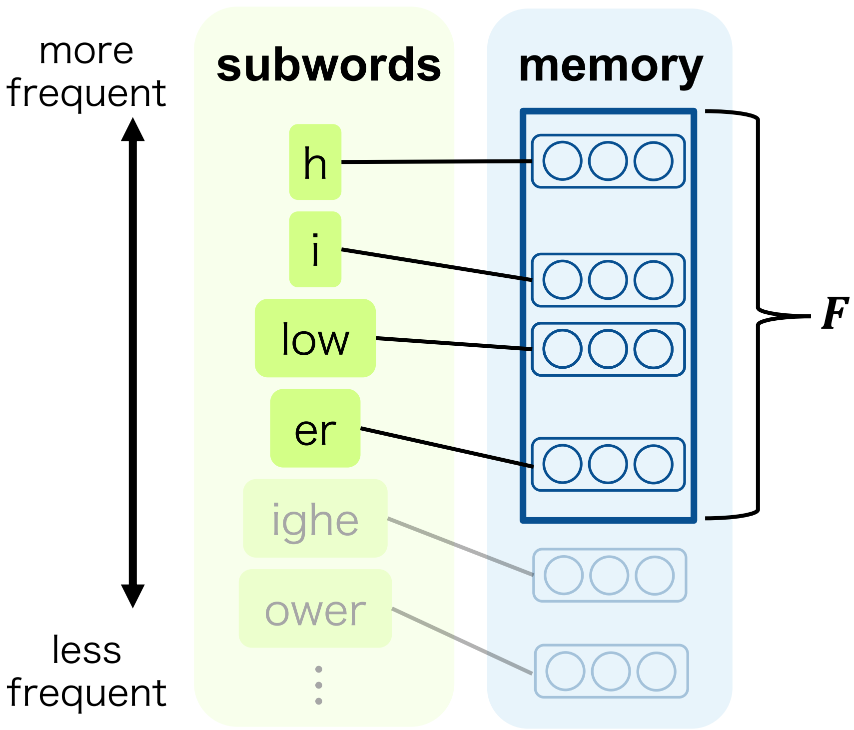
- Randomly share the same vectors between several subwords
- Model size $= H \times$ # of dimensions



Subword-to-Memory Mapping

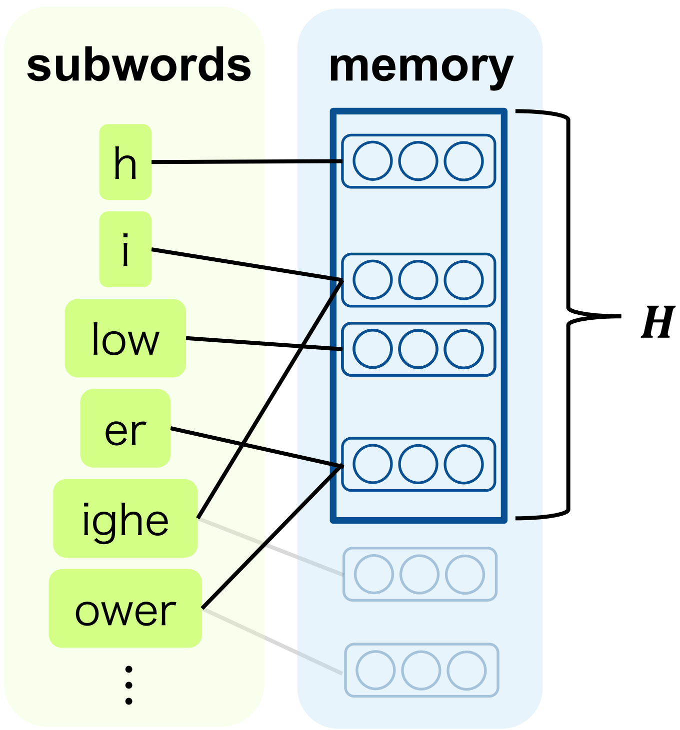
1. Discarding infrequent subwords

- Use top-F frequent subwords instead of all possible subwords
- Model size $= F \times$ # of dimensions



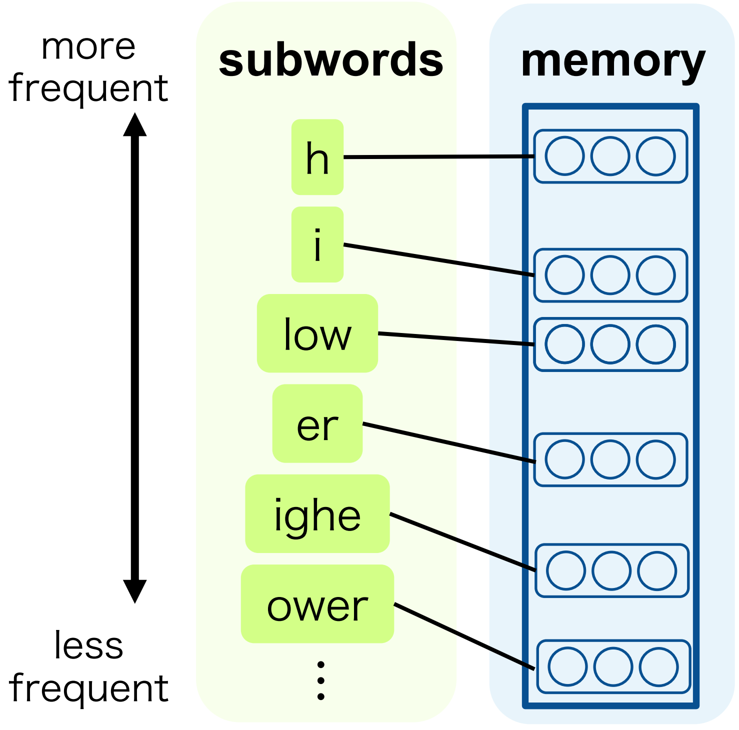
2. Memory sharing [Bojanowski, 2017]

- Randomly share the same vectors between several subwords
- Model size $= H \times$ # of dimensions



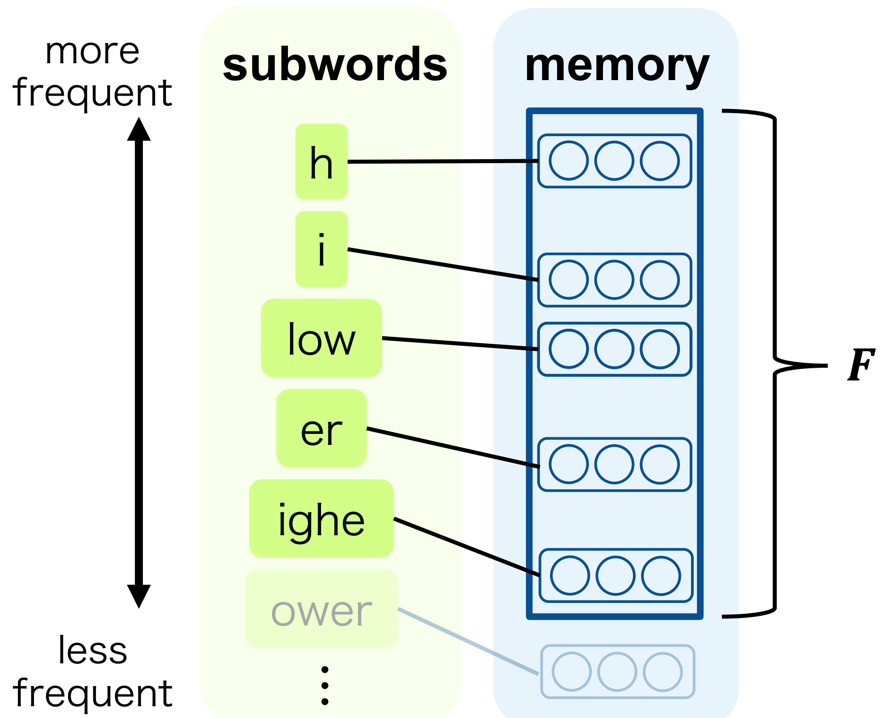
Subword-to-Memory Mapping

3. Combination
   I. Reduce subword vocabulary to top-F frequent subwords
   II. Apply memory sharing method



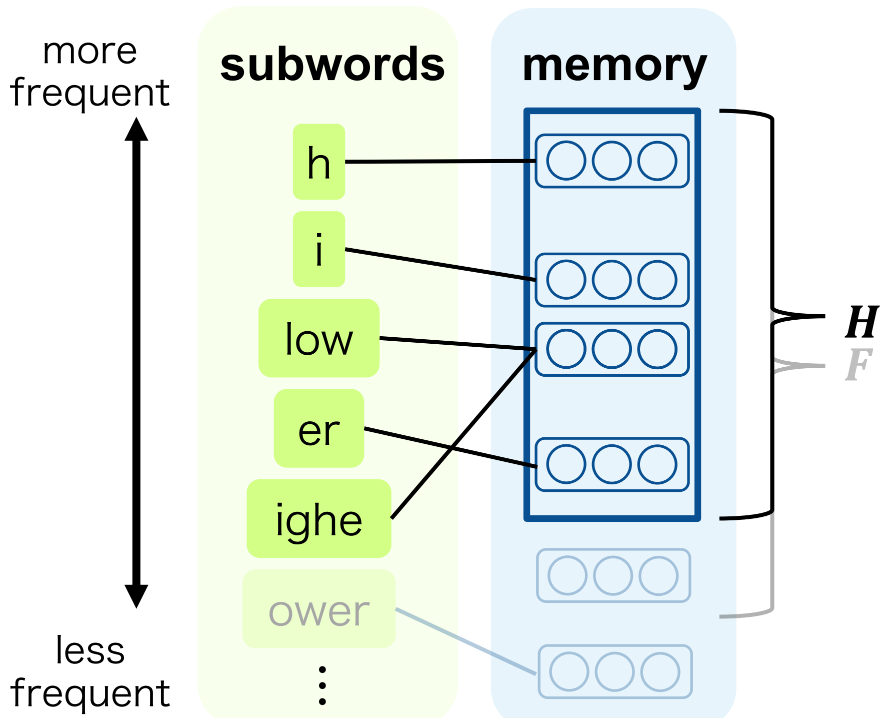
Subword-to-Memory Mapping

3. Combination
   I. Reduce subword vocabulary to top-F frequent subwords
   II. Apply memory sharing method



Subword-to-Memory Mapping

3. Combination
   I. Reduce subword vocabulary to top-F frequent subwords
   II. Apply memory sharing method



Key Ideas

- Subword-to-memory mapping function
  1. Discarding infrequent subwords
  2. Memory sharing
  3. Combination of 1. and 2.

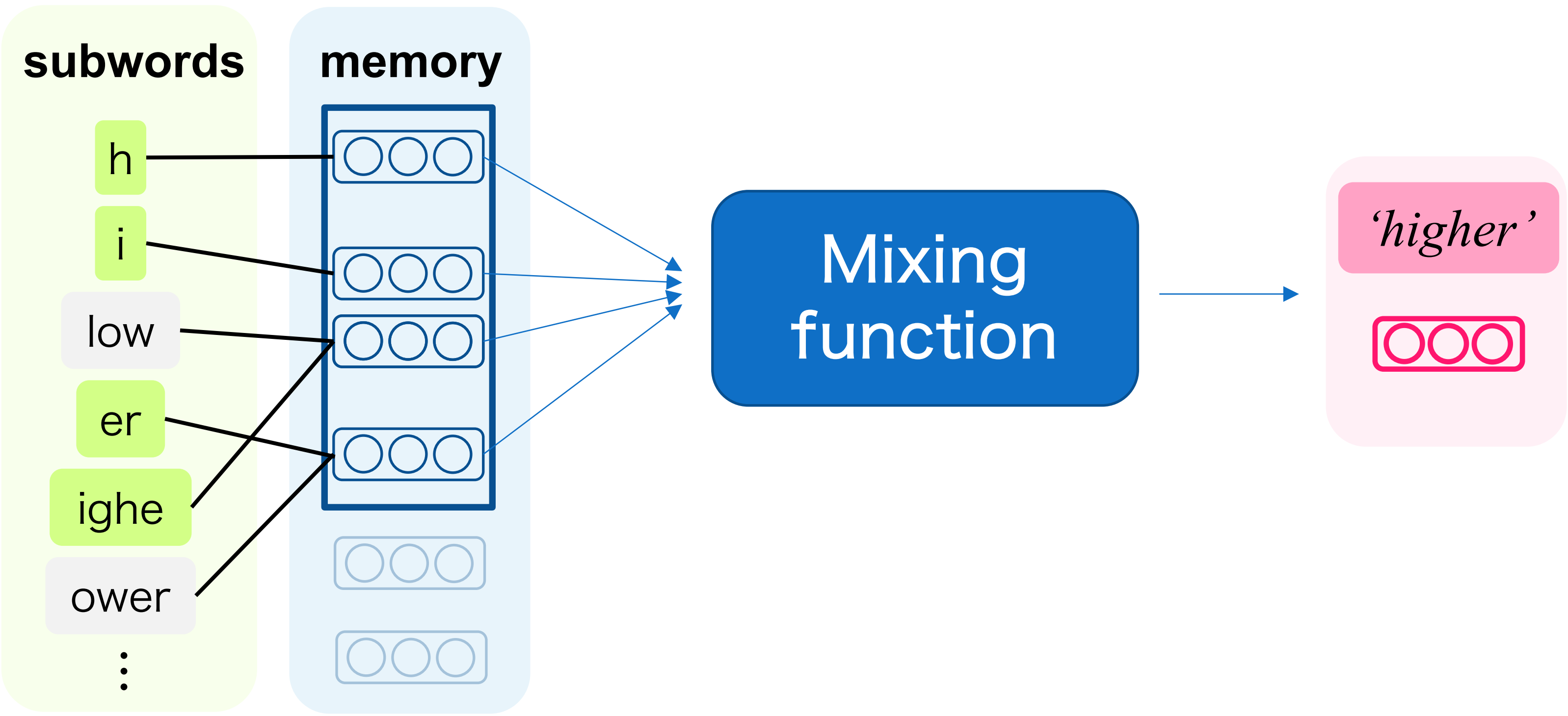
- Subword mixing function
  - Self-attention mechanism

Key Ideas

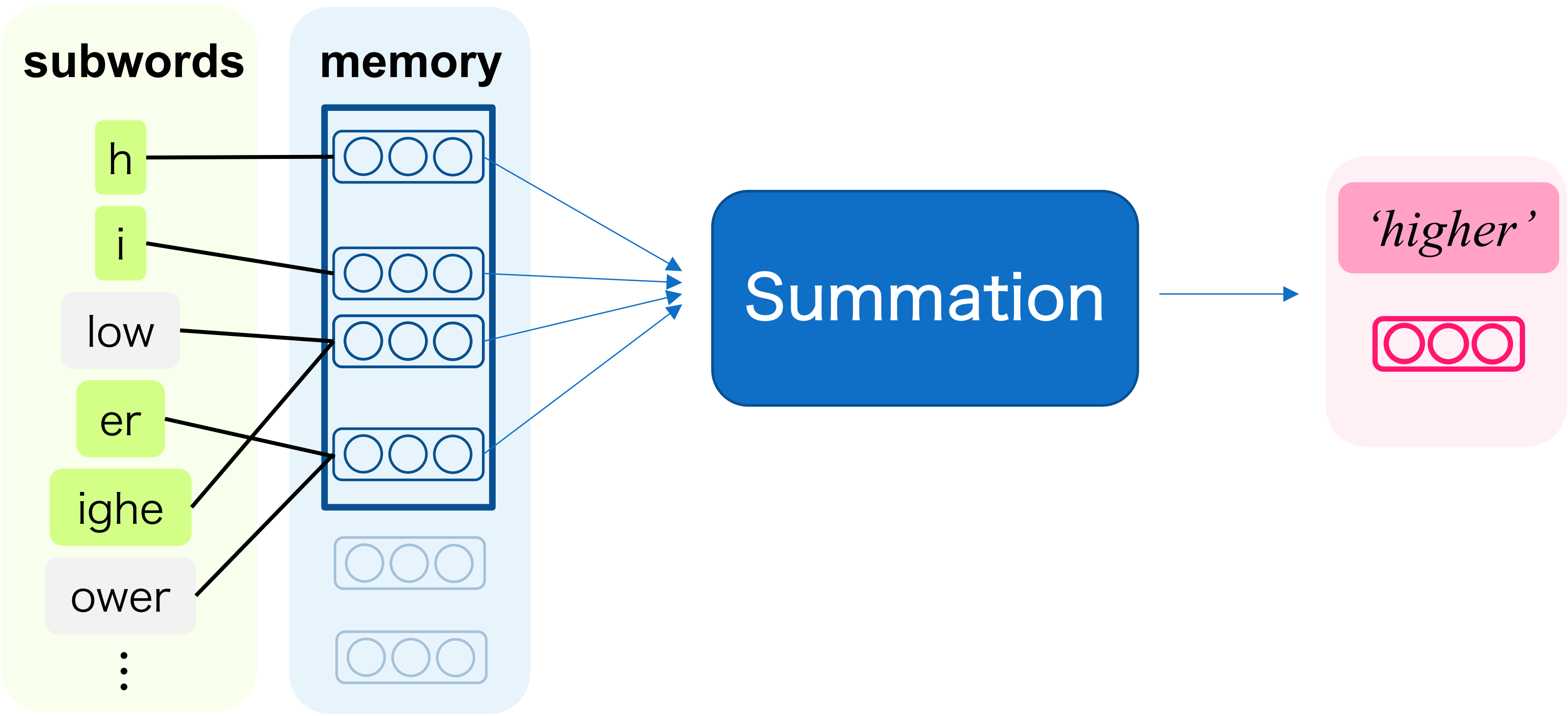
- Subword-to-memory mapping function
  1. Discarding infrequent subwords
  2. Memory sharing
  3. Combination of 1. and 2.

- Subword mixing function
  - Self-attention mechanism

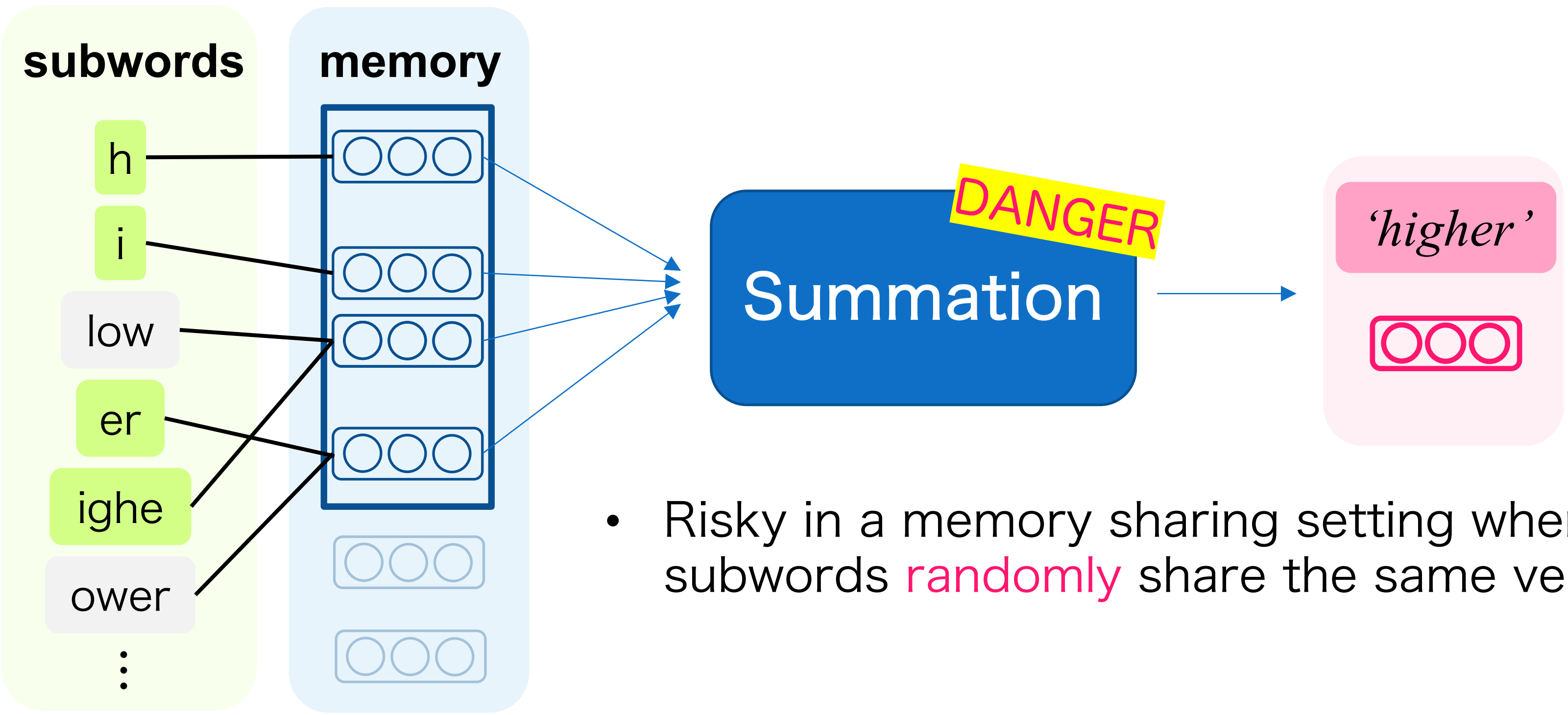
Subword Mixing Function



Subword Mixing Function

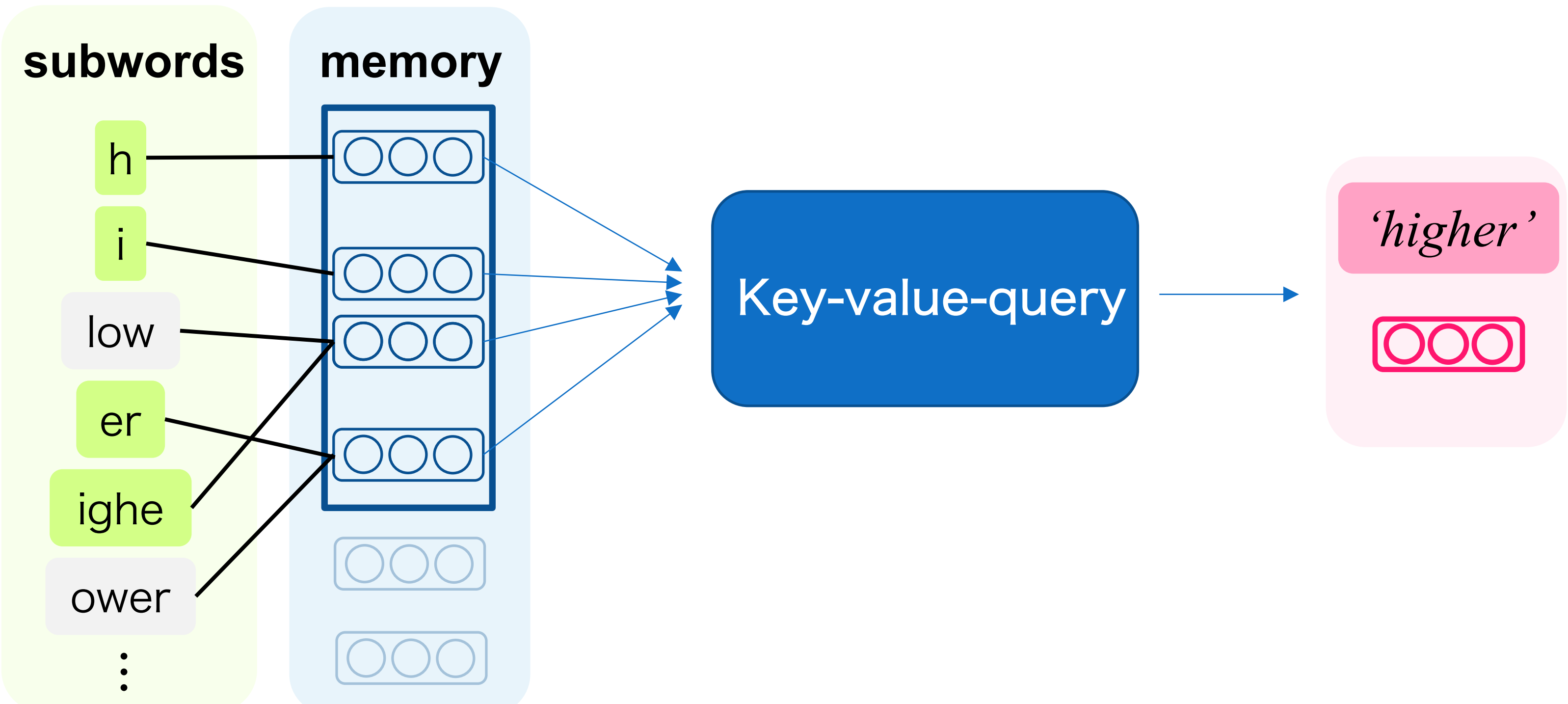


Subword Mixing Function



- Risky in a memory sharing setting where subwords randomly share the same vector

Subword Mixing Function



Subword Mixing Function

Key-value-query self-attention operation

- incorporate a "context-dependent" weighting factor $a_{s,w}$
- "context" = all the subwords obtained from word w

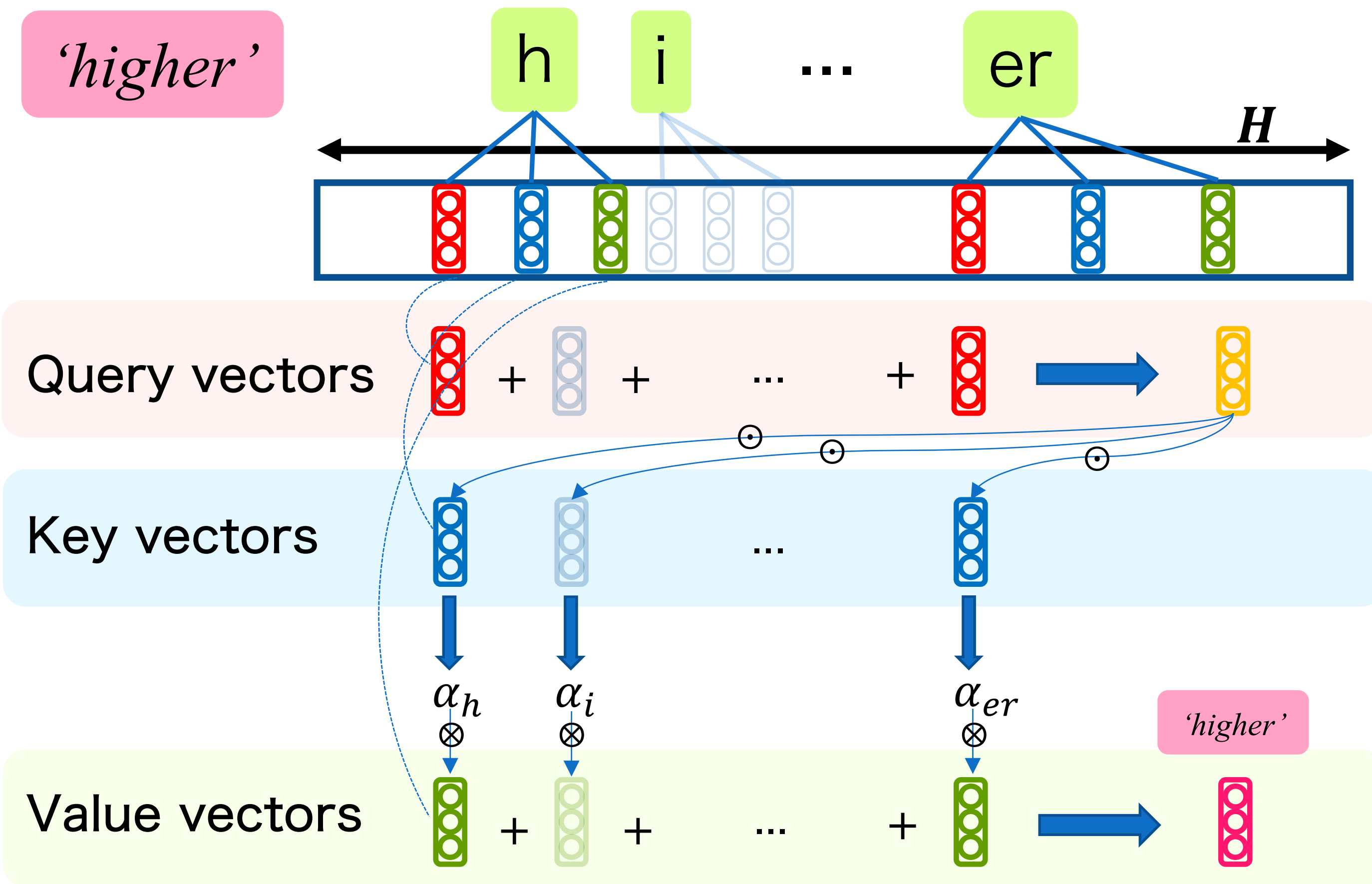
Summation

$$\tau_{\mathtt{sum}}(\boldsymbol{V}, w) = \sum_{s \in \phi(w)} \boldsymbol{v}_s.$$

Key-value-query

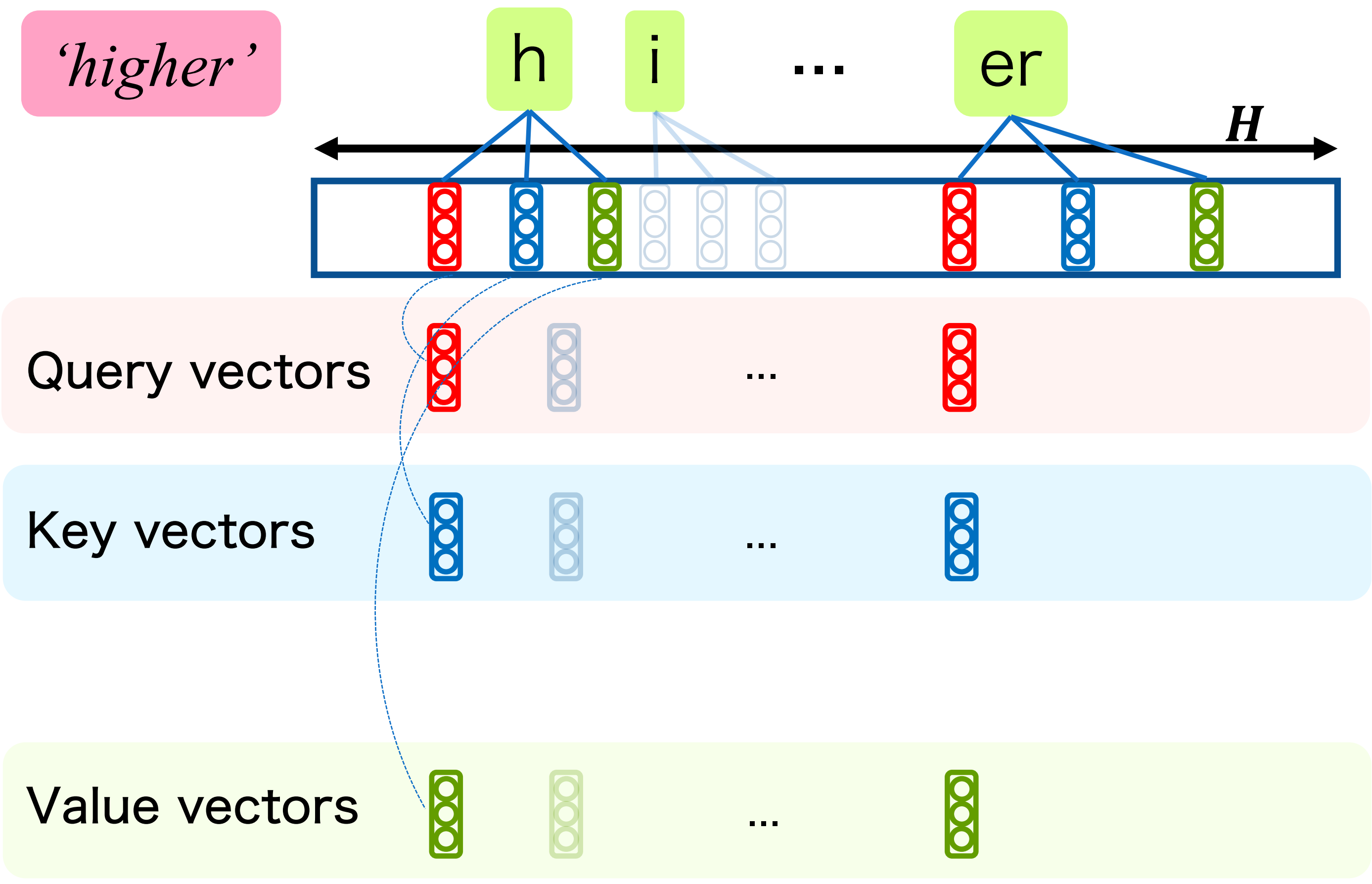
$$\tau_{\mathtt{kvq}}(\boldsymbol{V}, w) = \sum_{s \in \phi(w)} a_{s,w} \boldsymbol{v}_s$$

Modification of Mixing Function
- Key-value-query Operation

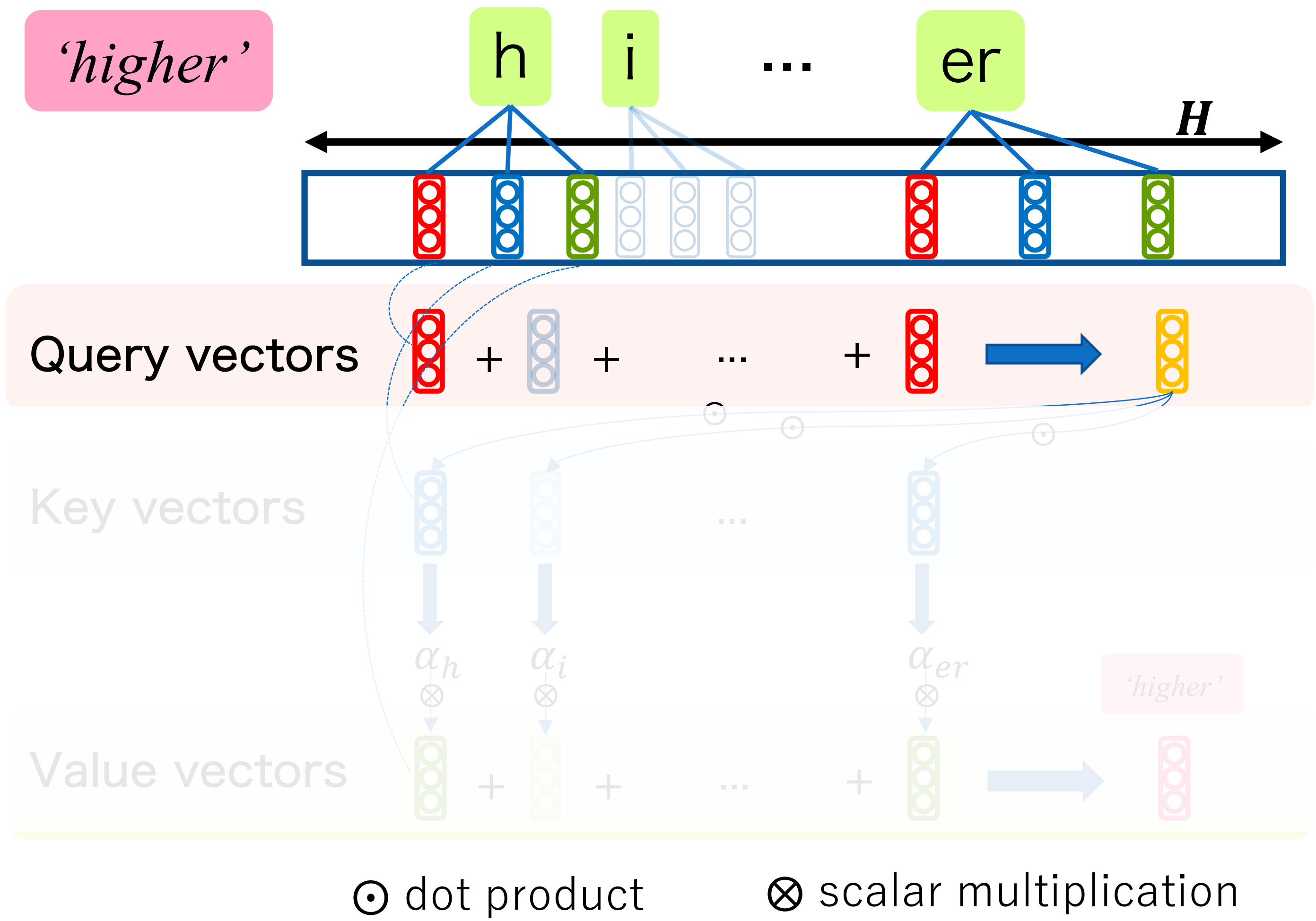


⊙ dot product ⊗ scalar multiplication

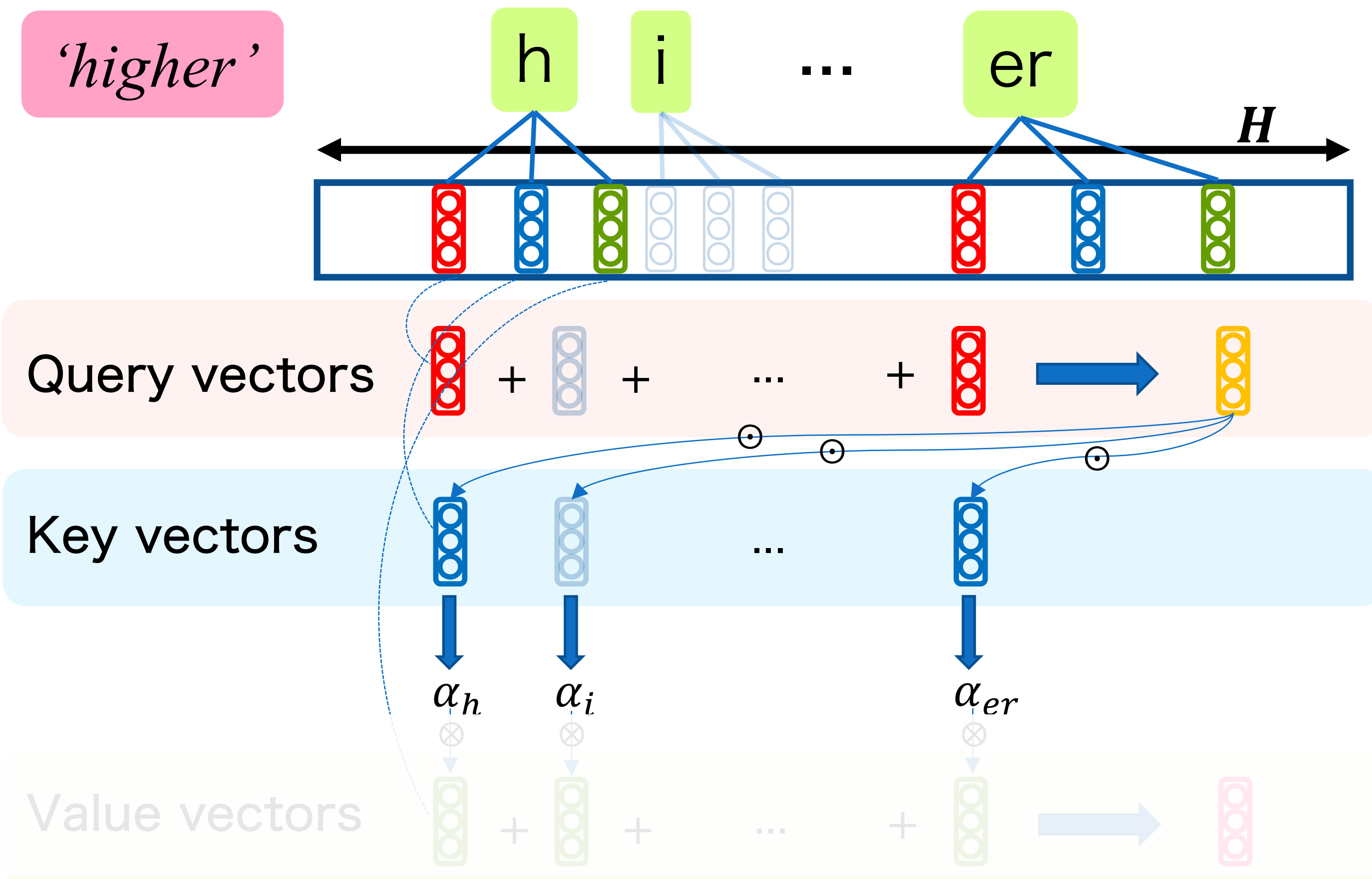
Modification of Mixing Function
- Key-value-query Operation



Modification of Mixing Function
- Key-value-query Operation

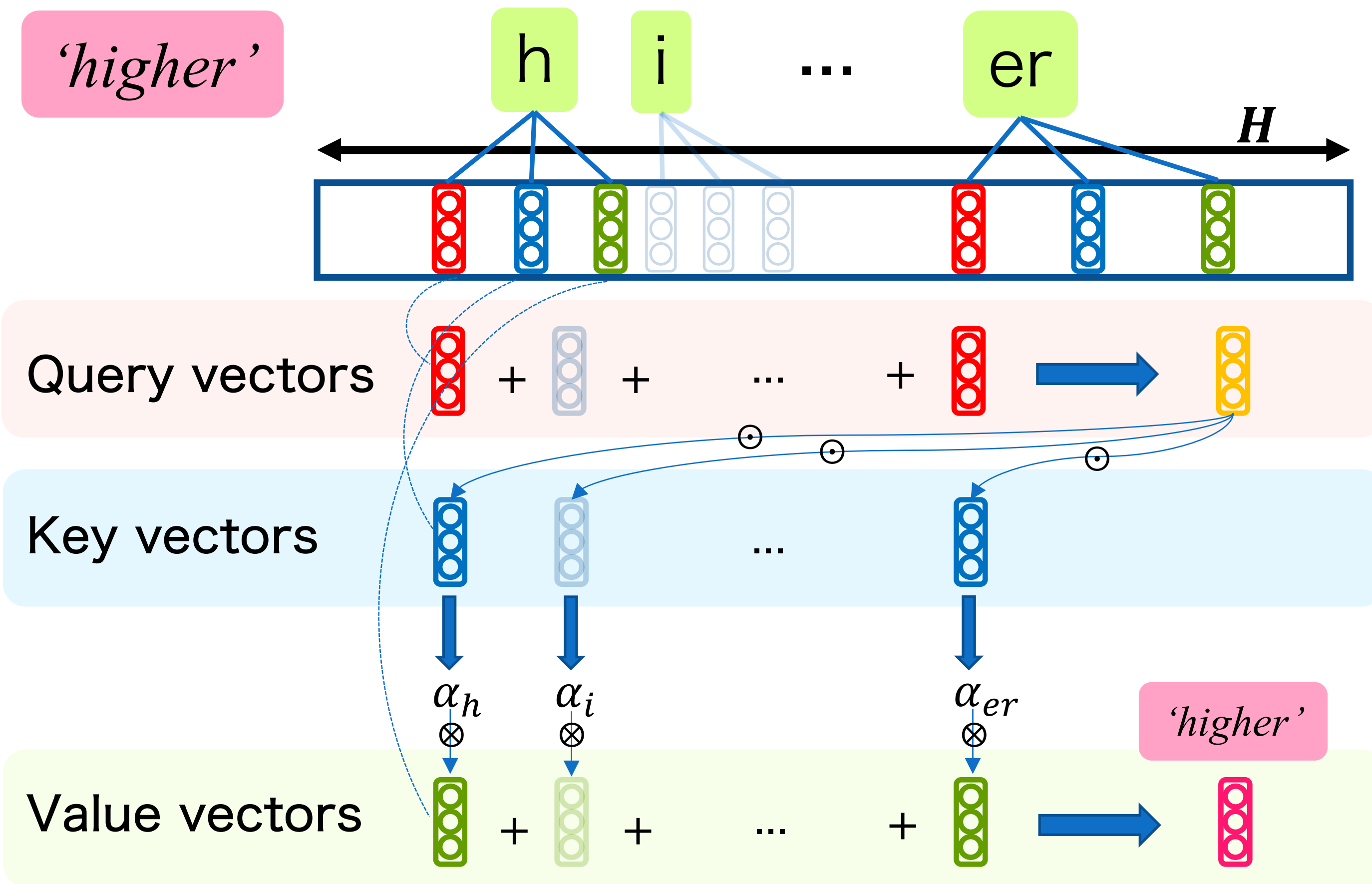


Modification of Mixing Function
- Key-value-query Operation



⊙ dot product ⊗ scalar multiplication

Modification of Mixing Function
- Key-value-query Operation



⊙ dot product ⊗ scalar multiplication

Modification of Mixing Function
- Key-value-query Operation

Advantages

- Highly expressive
  - allows assigning a lower weight to subword vector sharing its memory with completely unrelated subword
- No need of extra transformation matrix
  - Model size $= H \times$ # of dimensions

Key Ideas

- Subword-to-memory mapping function
  1. Discarding infrequent subwords
  2. Memory sharing
  3. Combination of 1. and 2.

- Subword mixing function
  - Self-attention mechanism

Outline

- Quick Overview
- Background
  - Word embeddings
  - Related work
  - Purpose
- Proposed Method
  - Key technique
    - Subword-to-memory mapping function
    - Subword mixing function
- Experiments
  - Word similarity with OOV words
  - Model compression test
  - Downstream tasks (NER, TE)

Evaluation of OOV Word Embeddings

- Word Similarity (Rare Word dataset)
  - Followed experimental settings used in [Zhao, EMNLP-2018]
  - 2000 word pairs, OOV rate : 11%

	method	Spearman's ρ
Baseline	Random	0.452
	BoS [Zhao, EMNLP-2018]	0.46*
Proposed	SUM-topF	0.513
	SUM-share ≈ rerun of BOS in our impl.	0.485
	KVQ-share	0.509
	SUM-comb	0.488
	KVQ-comb	**0.522**

✓ Our methods outperformed previous method

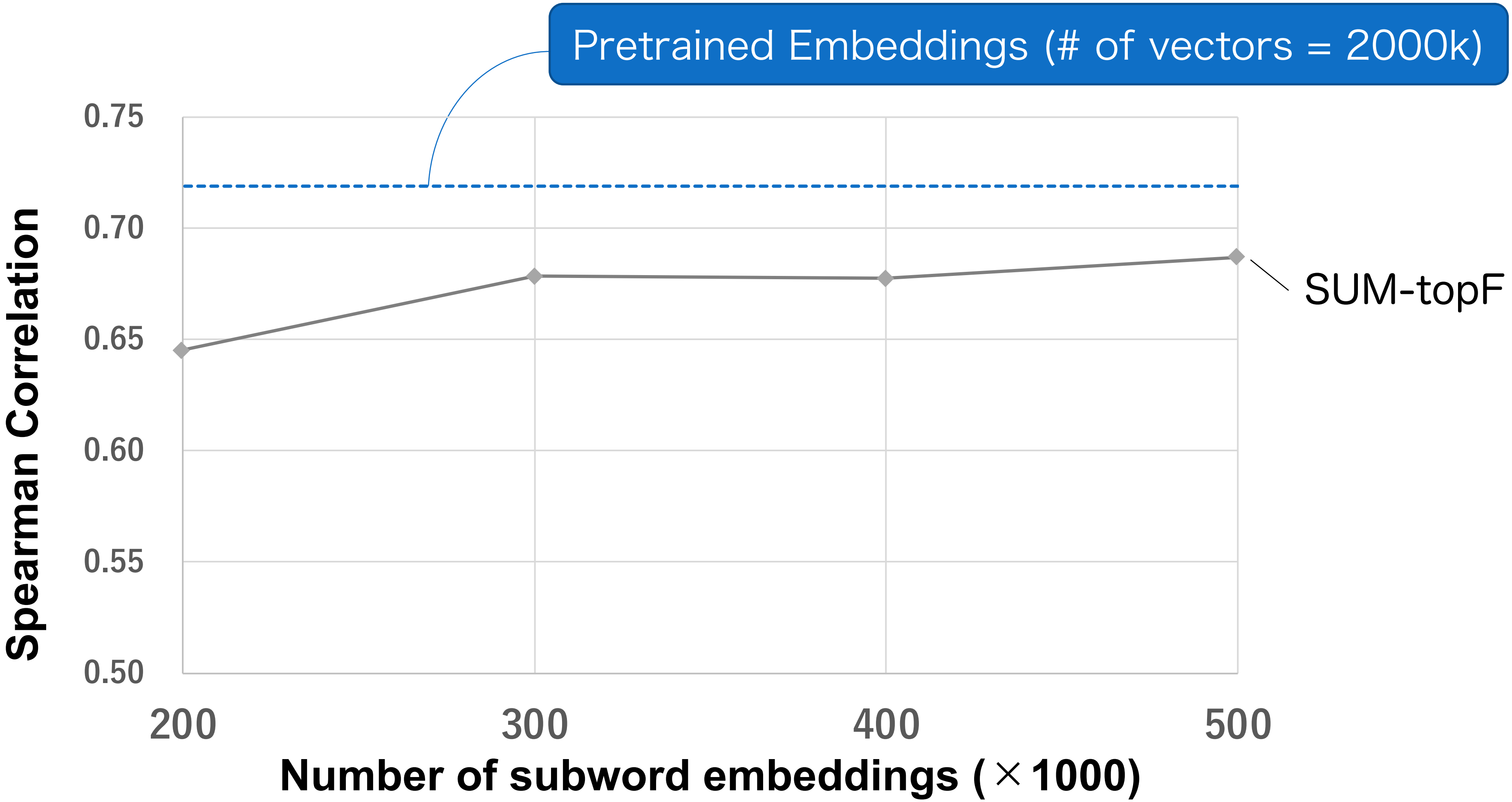
Evaluation of Model Compression

- **Evaluation tasks**
  - Word similarity task (9 datasets)

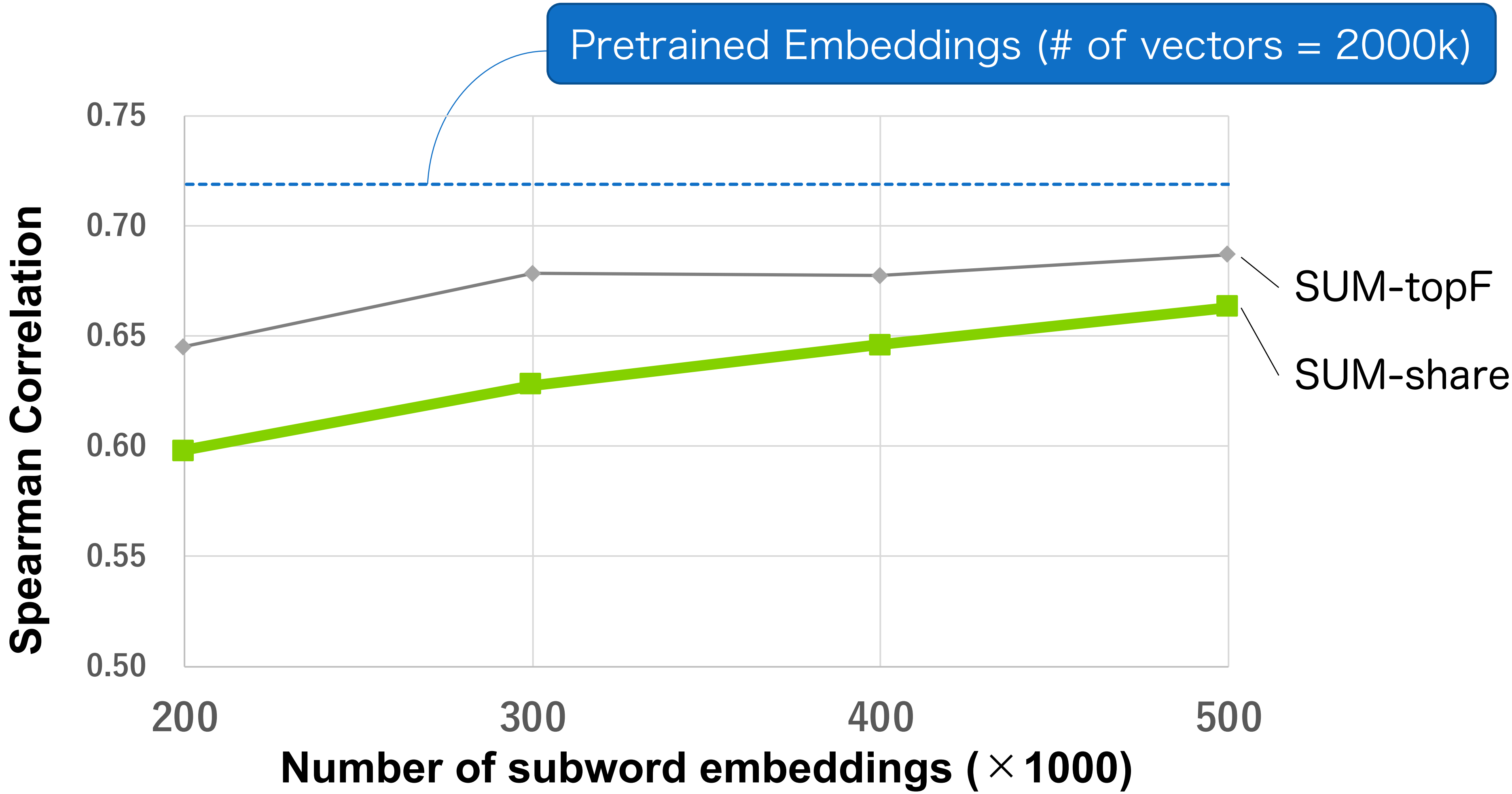
- **Pre-trained Embeddings**
  - fastText embeddings trained on Common Crawl corpus
    - 2M words, 300 dimensions

- **Note:** discarded pairs containing at least one OOV word

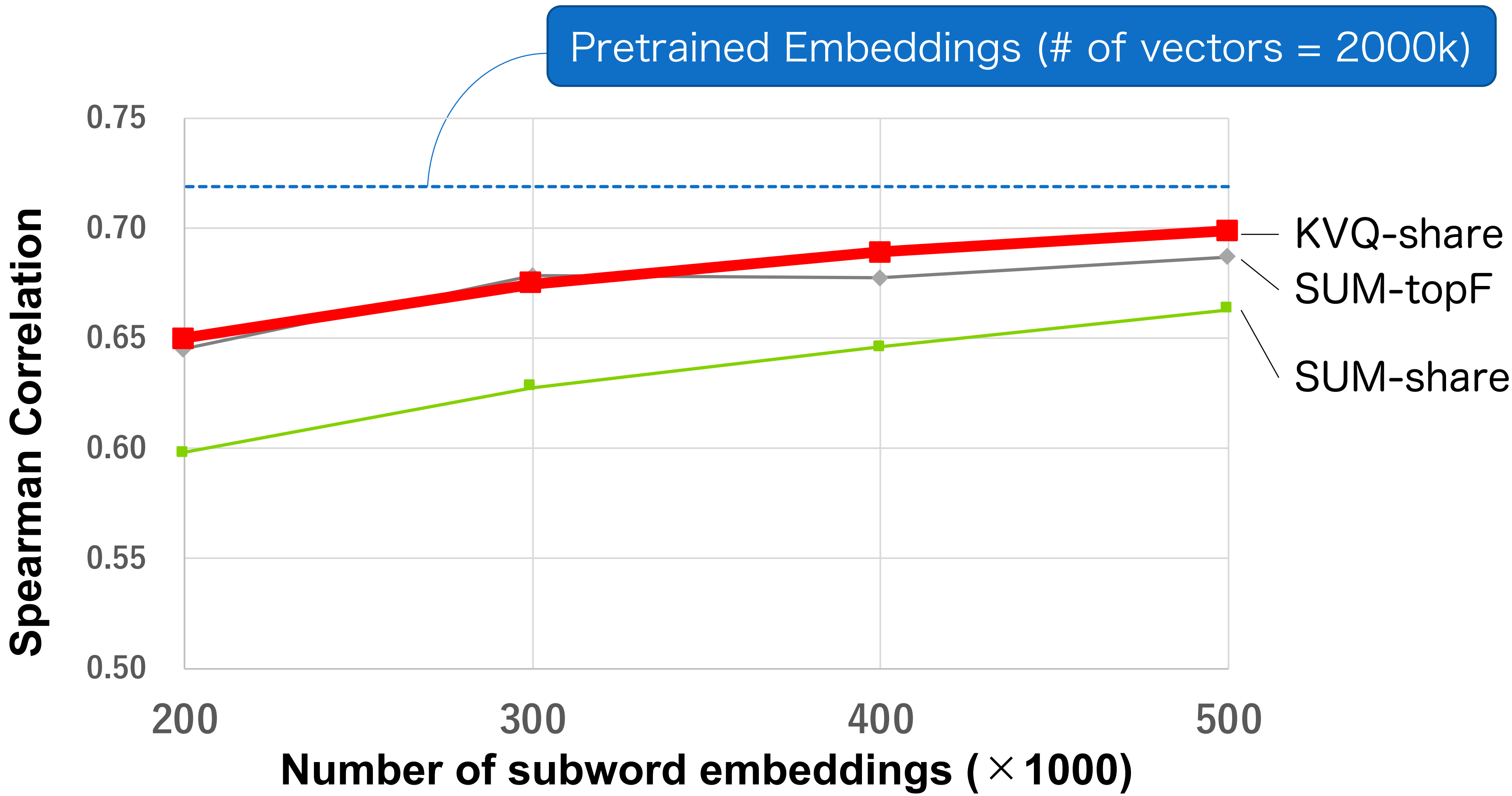
Results on Word Similarity Task



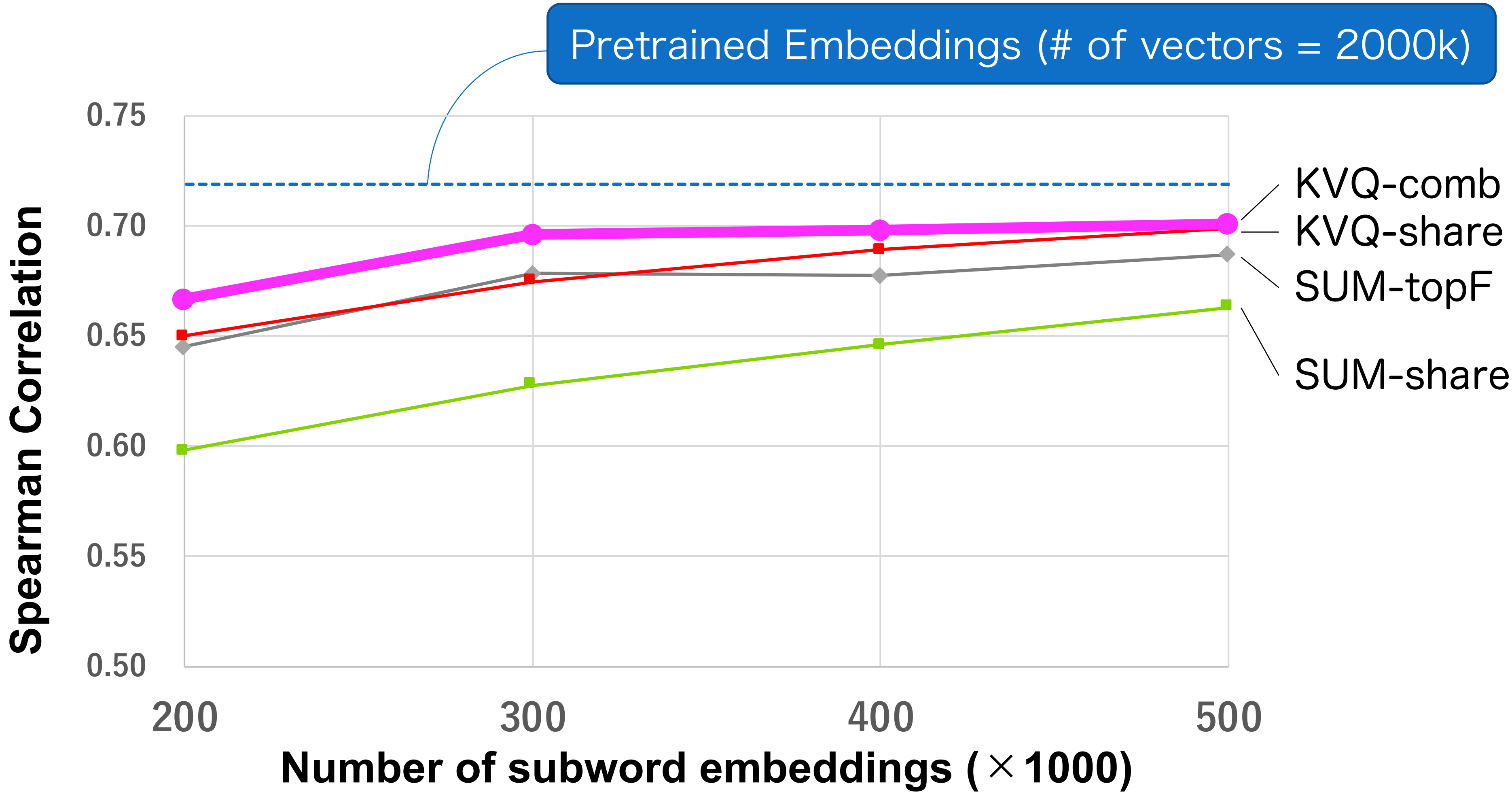
Results on Word Similarity Task



Results on Word Similarity Task



Results on Word Similarity Task



✓ KVQ-comb achieved comparable performance with less memory requirements

Evaluation on Downstream Tasks

- Used AllenNLP implementation, default settings

Textual Entailment (SNLI)

	Size (GB)	F1
fastText word emb.	2.23GB	87.8
KVQ-comb (H=0.5M)	0.59GB	88.0
KVQ-comb (H=0.2M)	0.23GB	87.6

Named Entity Recognition (CoNLL-2003)

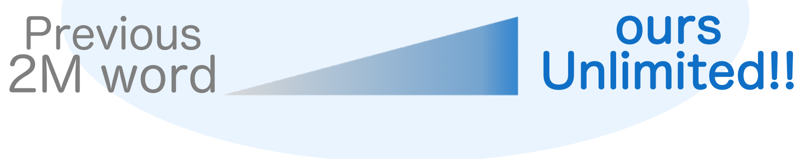
	Size (GB)	F1
fastText word emb.	2.23GB	90.3
KVQ-comb (H=0.5M)	0.59GB	90.4
KVQ-comb (H=0.2M)	0.23GB	89.3

✓ KVQ-comb achieved comparable performance with less memory requirements

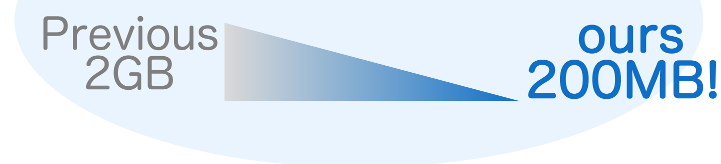
Conclusion

Proposal: novel word embeddings

OPEN-vocabulary

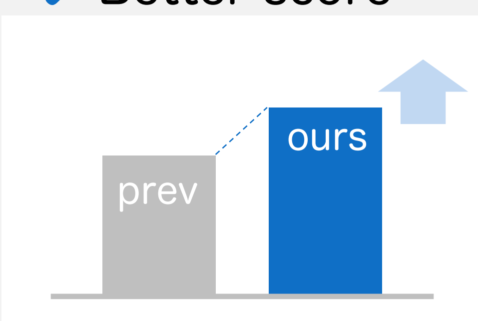


COMPACT model size



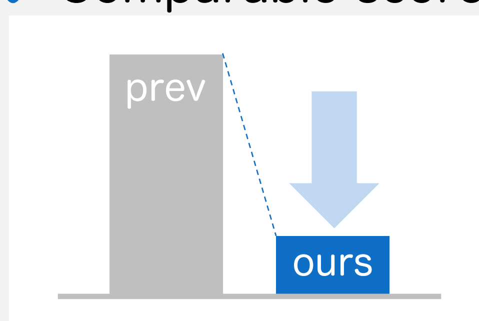
Performance:

✔ Better score



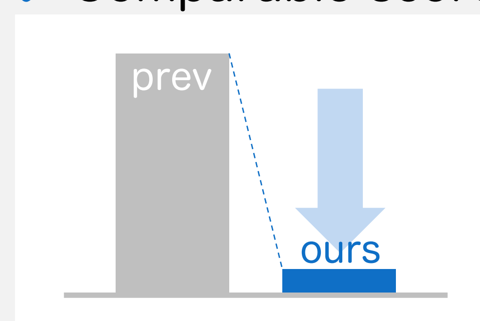
WordSim (OOV)

✔ 1/4 model size
✔ Comparable score



Model Compression Test

✔ 1/10 model size
✔ Comparable score



Downstream Tasks