EEP 596: LLMs: From Transformers to GPT || Lecture ~~8~~ 7

Dr. Karthik Mohan

Univ. of Washington, Seattle

January 26, 2026

Logistics

- Mini-Project 1 has two parts: Part 1 released last Friday and due coming Saturday
- MP1, Part 2 released this Tuesday and due two Fridays from now
- Do make use of discord and office hours, and review sessions for additional help from TAs and fellow students
- Paper Presentation Guidelines: Shared some on discord - Do take a look. Place yourself as someone new to the paper and listening to your presentation - Would they find it engaging?

Pointers for Paper Presentation

1. **Explain the key takeaway:** The key takeaways should be clearly understandable and accessible from the paper - Use examples if needed to make it accessible
2. **Motivate the paper:** Why was this paper needed in the first place? What problem is it solving - Explain it in simple words
3. **Use examples and figures:** To make your presentation accessible, use examples and figures in your slides
4. **Pick a paper you are comfortable presenting:** If its too technical and hard to simplify, then I suggest you pick a different paper that you can present with confidence!

Last Lecture

- BERT and Transformers Architecture
- Coding Exercise

Today's Lecture

- Multi-Head Attention

Deep Learning References

Deep Learning

Great reference for the theory and fundamentals of deep learning: Book by Goodfellow and Bengio et al Bengio et al

Deep Learning History

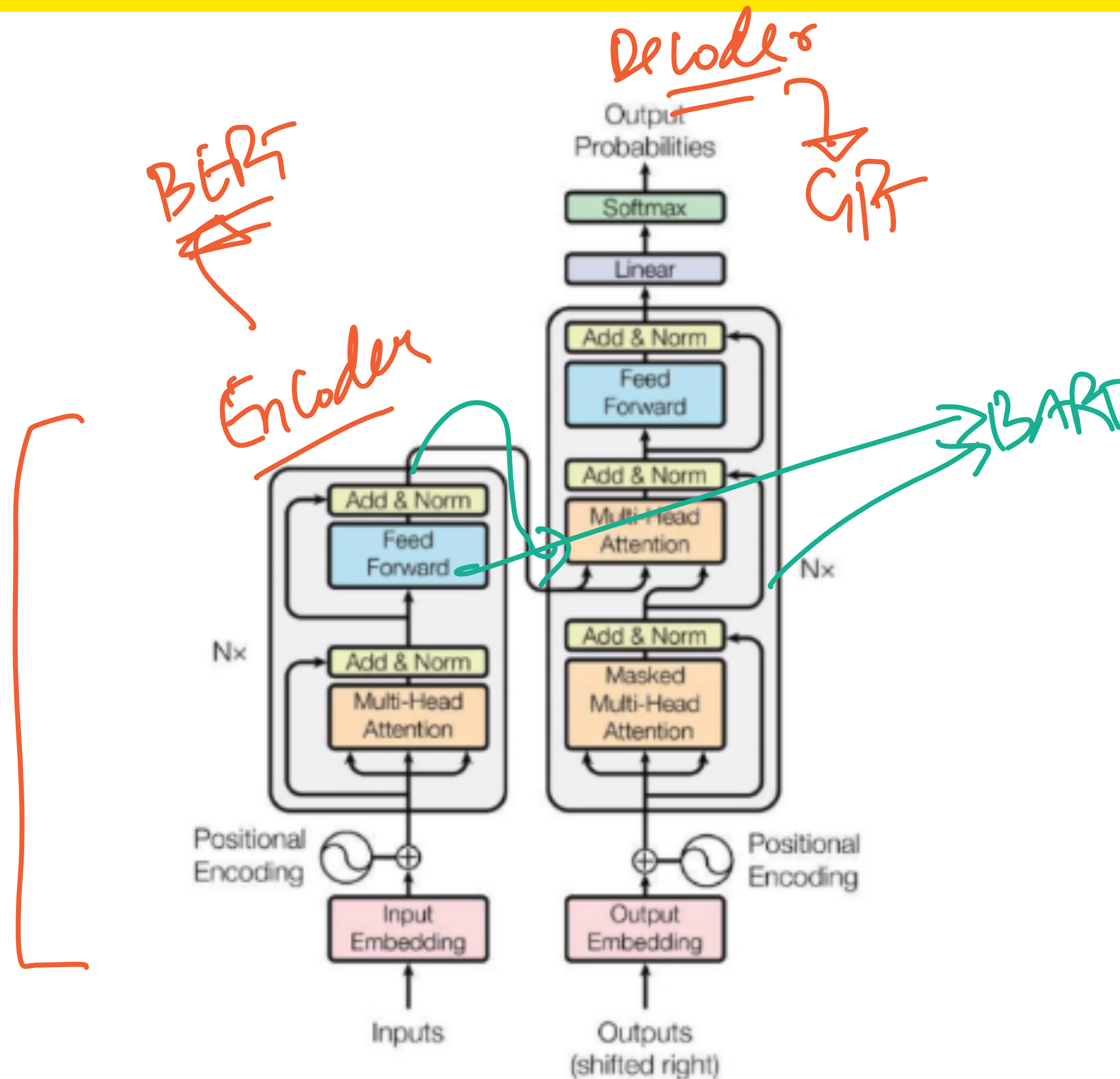
Embeddings

SBERT and its usefulness SBert Details Instacart Search Relevance Instacart Auto-Complete

Attention

Illustration of attention mechanism

Transformers - Encoder and Decoder Architecture



Path of a sentence through transformer layers

1️⃣ Raw Input Sentence

Example input:

> **"I love machine learning"**

At this stage, it's just a string—no math yet.

Path of a sentence through transformer layers

2️⃣ Text Normalization (Basic Preprocessing)

Before tokenization, BERT applies **basic text cleaning**:

- Lowercasing *(for uncased models)*
- Unicode normalization
- Whitespace cleanup

Result:

```arduino
"i love machine learning"
```

Path of a sentence through transformer layers

3️⃣ WordPiece Tokenization

BERT uses **WordPiece tokenization**, which breaks words into subword units.

Steps:

1. Split on whitespace
2. Decompose unknown or rare words into subwords
3. Prefix continuation pieces with `##`

Example:

```css
["i", "love", "machine", "learning"]
```

If a word were rare:

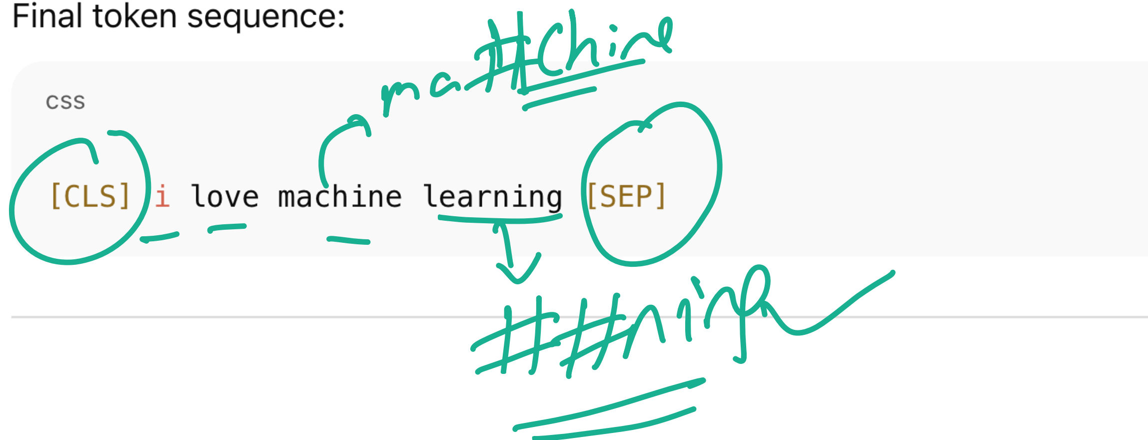
```arduino
"unhappiness" → ["un", "##happi", "##ness"]
```

Path of a sentence through transformer layers

4️⃣ Add Special Tokens

BERT **requires special tokens** for context and classification.

Token	Purpose
`[CLS]`	Sentence-level representation
`[SEP]`	Sentence separator / end marker



Path of a sentence through transformer layers

5️⃣ Convert Tokens → Token IDs

Each token is mapped to an integer via BERT's vocabulary.

Example (IDs are illustrative):

```yaml
[101, 1045, 2293, 3698, 4083, 102]
```

CLS SEP

Now the sentence is numerical.

6️⃣ Create Input Embeddings (Critical Step)

Each token gets **3 embeddings**, which are **summed element-wise**:

a) Token Embeddings

Learned vector for each word/subword.

b) Position Embeddings

Encode token order:

```csharp
[CLS]=pos0, i=pos1, love=pos2, ...
```

c) Segment (Token Type) Embeddings

Used to distinguish sentence A vs B.

- Single sentence → all zeros

Final embedding per token:

```ini
Embedding = Token + Position + Segment
```

Path of a sentence through transformer layers

7️⃣ Input Matrix to Transformer

At this point, your sentence is a matrix:

```rust
(sequence_length × hidden_size)
= (6 × 768)    ← for BERT-Base
```

Each row corresponds to one token.

Path of a sentence through transformer layers

8️⃣ Pass Through Transformer Encoder Layers

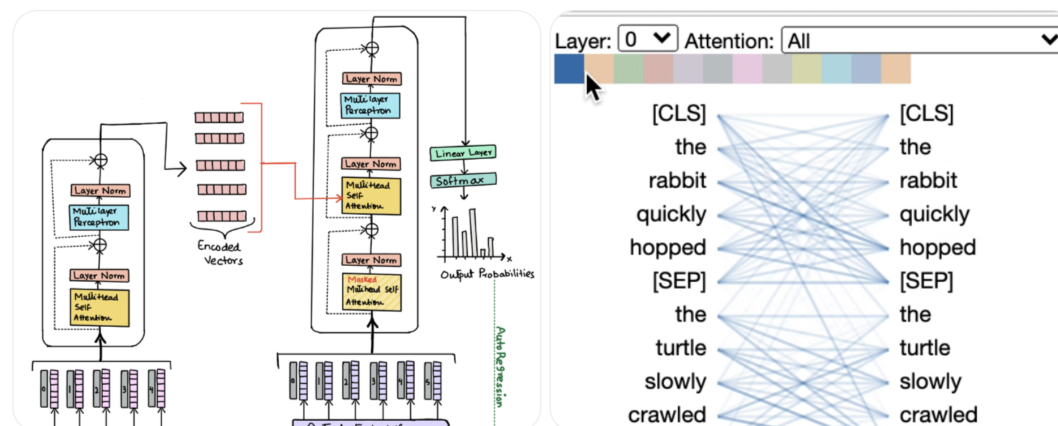
BERT-Base has **12 identical Transformer layers.**

Each layer contains:

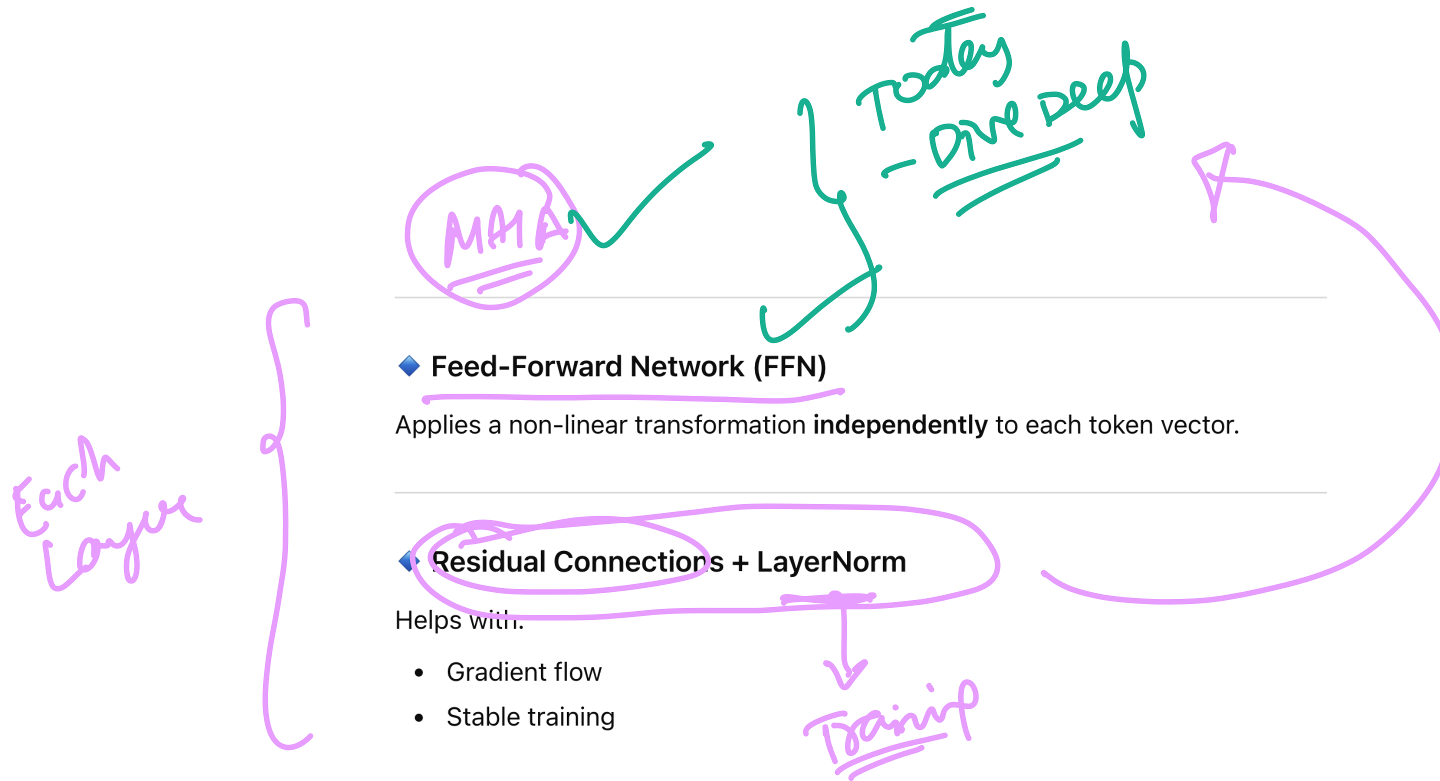
🔷 Multi-Head Self-Attention

Every token attends to **every other token** (including itself).

This lets `[CLS]` gather information from the entire sentence.



Path of a sentence through transformer layers



Path of a sentence through transformer layers

🔟 Extract the `[CLS]` Token

The **first vector** corresponds to `[CLS]`:

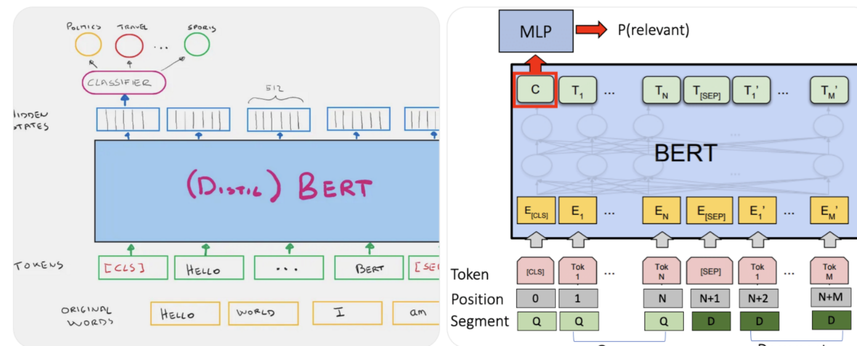
```ini
CLS_embedding = last_hidden_state[:, 0, :]
```

Shape:

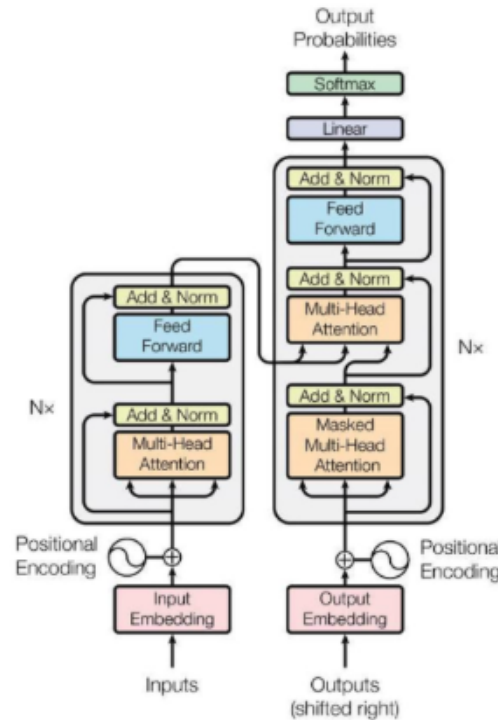
```scss
(768,)
```

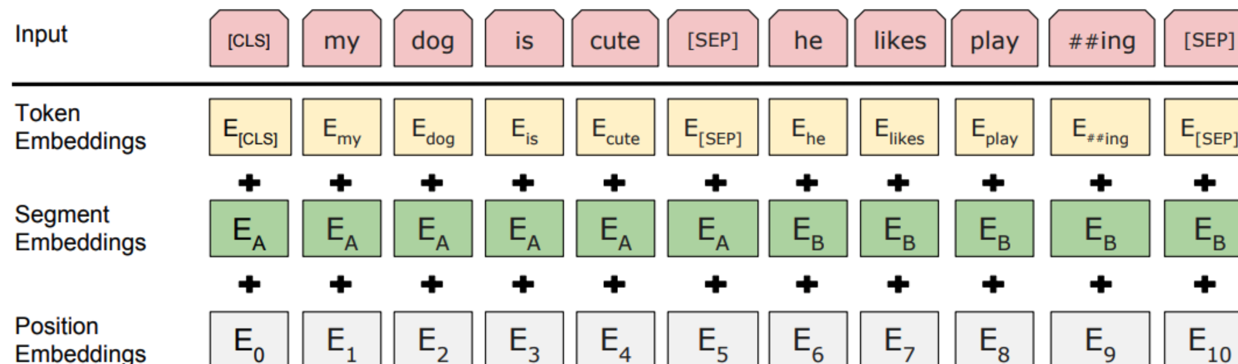
This is the **sentence embedding** used for:

- Classification
- Regression
- Sentence-level tasks



Understanding Encoder/BERT at high-level





Transformer Types in Practice

Encoder Only

- BERT, SBERT, ViTransformer, etc
- Uses only self-attention and FFN blocks
- Good for classification, summarization, intent detection, image embeddings, etc

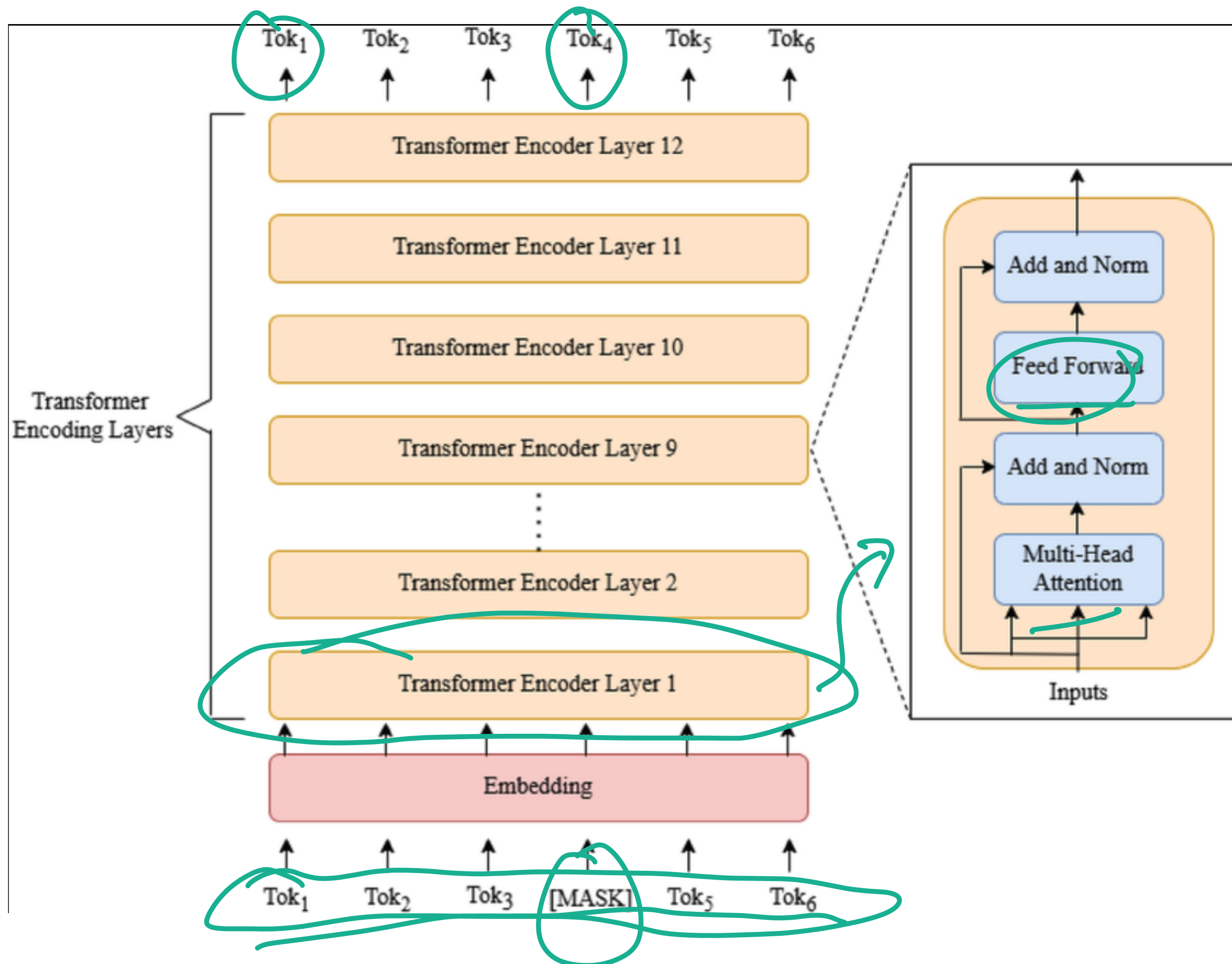
Encoder-Decoder

T5, BART. Also uses encoder-decoder attention

Decoder Only

- GPT, Llama, DeepSeek, etc
- Good for many tasks including encoder-only tasks and also generation tasks
- Uses self-attention and FFN blocks

BERT Encoding Layers



Transformation through Self-Attention and FFN

Setup (Before Layer 0)

Tokens

```css
["thinking", "machines"]
```

Embedding size

- d_model = 768
- Number of tokens T = 2

Transformation through Self-Attention and FFN

1️⃣ Input embeddings (Layer 0 input)

Each token gets:

- **Token embedding**
- **Position embedding**
- **Segment embedding**

They are **summed**:

$$x_i^{(0)} = E_{\text{token}}(i) + E_{\text{position}}(i) + E_{\text{segment}}(i)$$

So we have:

$$X^{(0)} = \begin{bmatrix} x_{\text{thinking}} \\ x_{\text{machines}} \end{bmatrix} \in \mathbb{R}^{2\times768}$$

This matrix is the **input to Layer 0**.

768

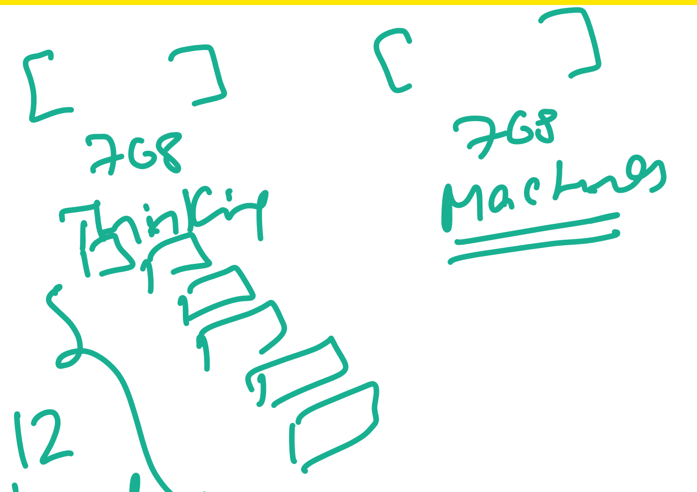
768

ICE #1

Say token embedding for thinking is $[1, 2]$ and for machines is $[-1, 3]$. Let the position embedding for position 1 be $[2, 3]$ and for position 2 be $[1, 5]$. Assume there is no segment embedding or set it to $[0, 0]$. What is the input embedding vector for machines?

1. $[0, 8]$
2. $[1, 6]$
3. $[3, 5]$
4. $[2, 7]$

Transformation through Self-Attention and FFN



2 Self-Attention: Create Q, K, V

BERT uses **multi-head attention.**

- Heads = 12
- Head dimension = $d_k = 768/12 = 64$

For **each head h**, we learn matrices:

$$W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in \mathbb{R}^{768 \times 64}$$

Compute Q, K, V

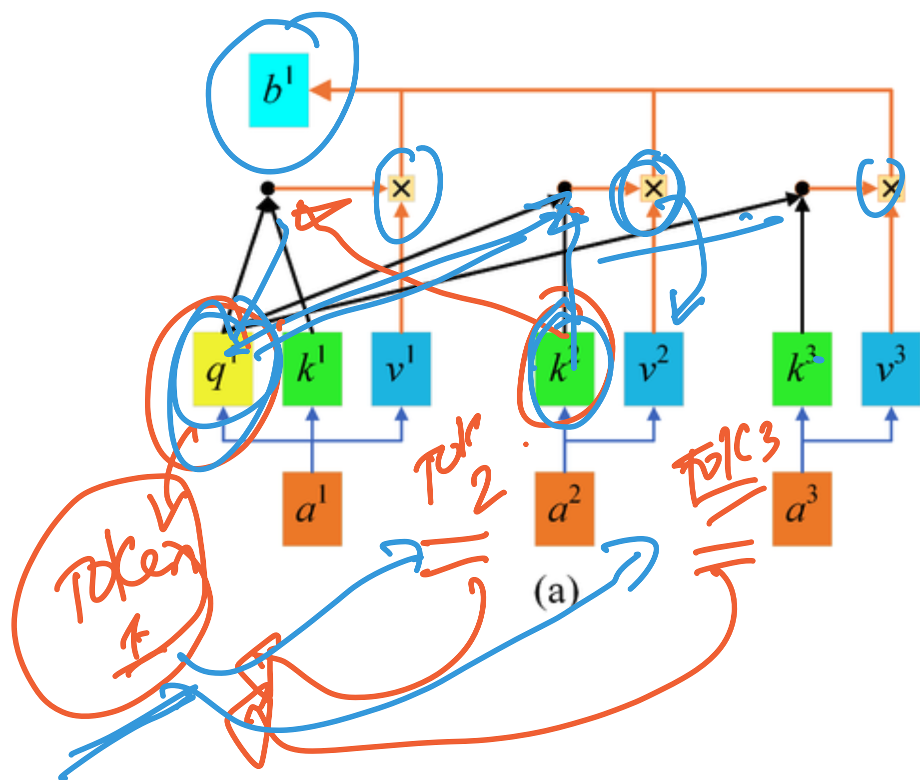
For both tokens:

$$Q = X^{(0)} W_Q \quad K = X^{(0)} W_K \quad V = X^{(0)} W_V$$

Each is:

$$Q, K, V \in \mathbb{R}^{2 \times 64}$$

Transformation through Self-Attention and FFN



(a)

$q^i = W^q a^i$ $\quad [q^1\ q^2\ q^3] = W^q\,[a^1\ a^2\ a^3]$ (Q)

$k^i = W^k a^i$ $\quad [k^1\ k^2\ k^3] = W^k\,[a^1\ a^2\ a^3]$ (K)

$v^i = W^v a^i$ $\quad [v^1\ v^2\ v^3] = W^v\,[a^1\ a^2\ a^3]$ (V)

(b)

Transformation through Self-Attention and FFN

3️⃣ Attention scores (who looks at whom)

Compute **scaled dot-product attention:**

$$A = \frac{QK^\top}{\sqrt{d_k}}$$

So:

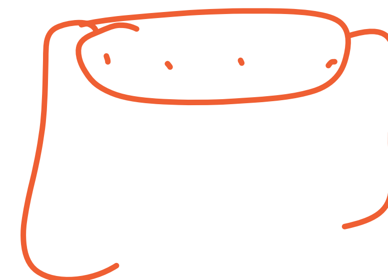
$$A \in \mathbb{R}^{2\times 2}$$

Example interpretation:

```ini
A =
[ thinking→thinking   thinking→machines
  machines→thinking   machines→machines ]
```

Each value = *similarity of queries to keys*.

Transformation through Self-Attention and FFN

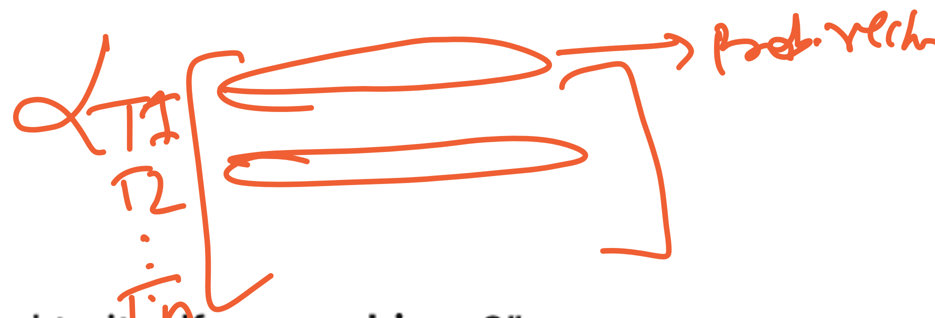


4️⃣ Softmax (turn scores into weights)

Row-wise softmax:

$$\alpha_{ij} = \frac{e^{A_{ij}}}{\sum_j e^{A_{ij}}}$$

Now each row sums to 1.



This answers:

> "How much should **thinking** attend to itself vs **machines?**"

Transformation through Self-Attention and FFN

5 **Weighted sum of values**

$$Z = \alpha V$$

Result:

$$Z \in \mathbb{R}^{2\times 64}$$

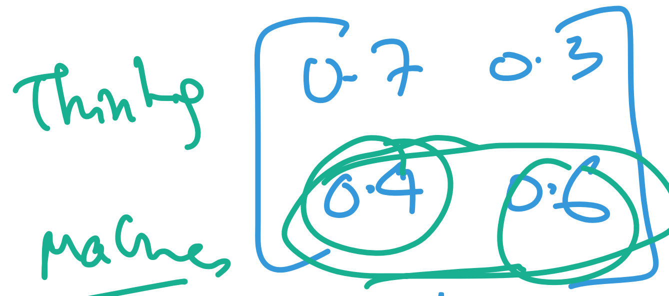
Meaning:

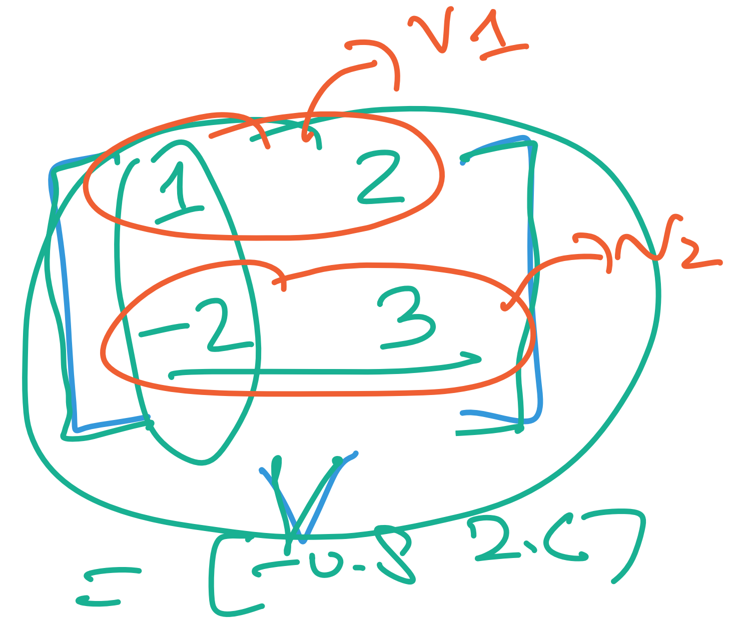
- "thinking" becomes a **contextual blend** of thinking + machines
- same for "machines"

ICE #2

Assume $v1 = [1, 2]$, $v2 = [-2, 3]$ and $\alpha_{11} = 0.7, \alpha_{12} = 0.3, \alpha_{21} = 0.4, \alpha_{22} = 0.6$. What is the transformed token embedding for the token 'Thinking' for this particular head, post self-attention?

1. $[1.3, 2]$
2. $[-0.8, 2.6]$





Transformation through Self-Attention and FFN

6 Concatenate heads + output projection

After doing this for **all 12 heads**:

$$Z_{\text{concat}} \in \mathbb{R}^{2\times 768}$$

Final projection:

$$Z_{\text{attn}} = Z_{\text{concat}} W_O \quad W_O \in \mathbb{R}^{768\times 768}$$

Transformation through Self-Attention and FFN

7️⃣ **Residual connection + LayerNorm**

$$\tilde{X} = \mathrm{LayerNorm}(X^{(0)} + Z_{\mathrm{attn}})$$

This stabilizes training and preserves original info.

🔶 Feed-Forward Network (FFN)

Now each token is processed **independently** (no cross-token mixing here).

Transformation through Self-Attention and FFN

🔶 Feed-Forward Network (FFN)

Now each token is processed **independently** (no cross-token mixing here).

8️⃣ First linear layer (expansion)

$$H = \tilde{X}W_1 + b_1$$

Where:

- $W_1 \in \mathbb{R}^{768\times 3072}$

Result:

$$H \in \mathbb{R}^{2\times 3072}$$

9️⃣ GELU activation

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

This introduces **nonlinearity** (soft gating).

Transformation through Self-Attention and FFN

🔟 Second linear layer (compression)

$$F = \mathrm{GELU}(H)W_2 + b_2$$

Where:

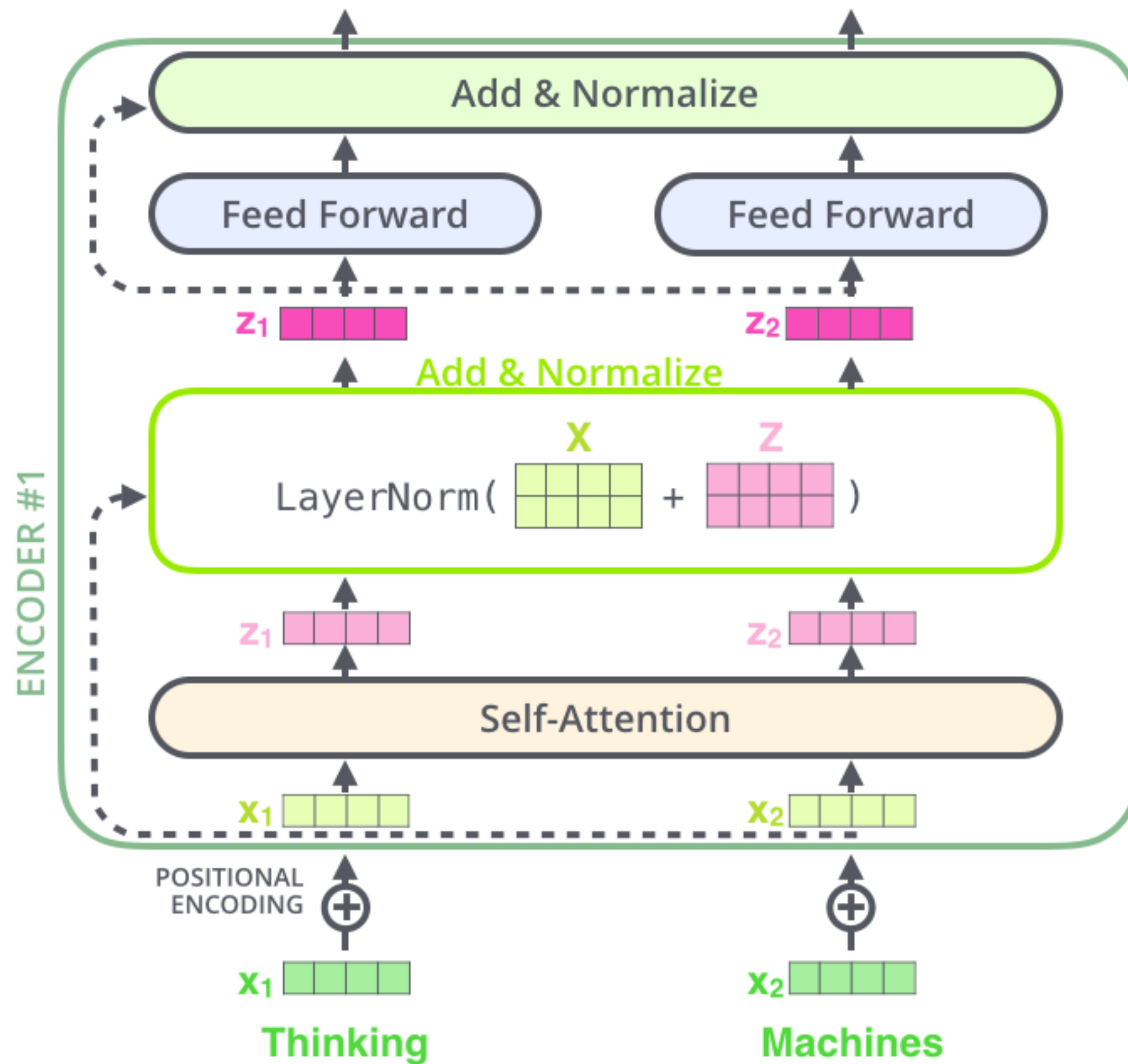
- $W_2 \in \mathbb{R}^{3072 \times 768}$

1️⃣ 1️⃣ Residual + LayerNorm (end of layer)

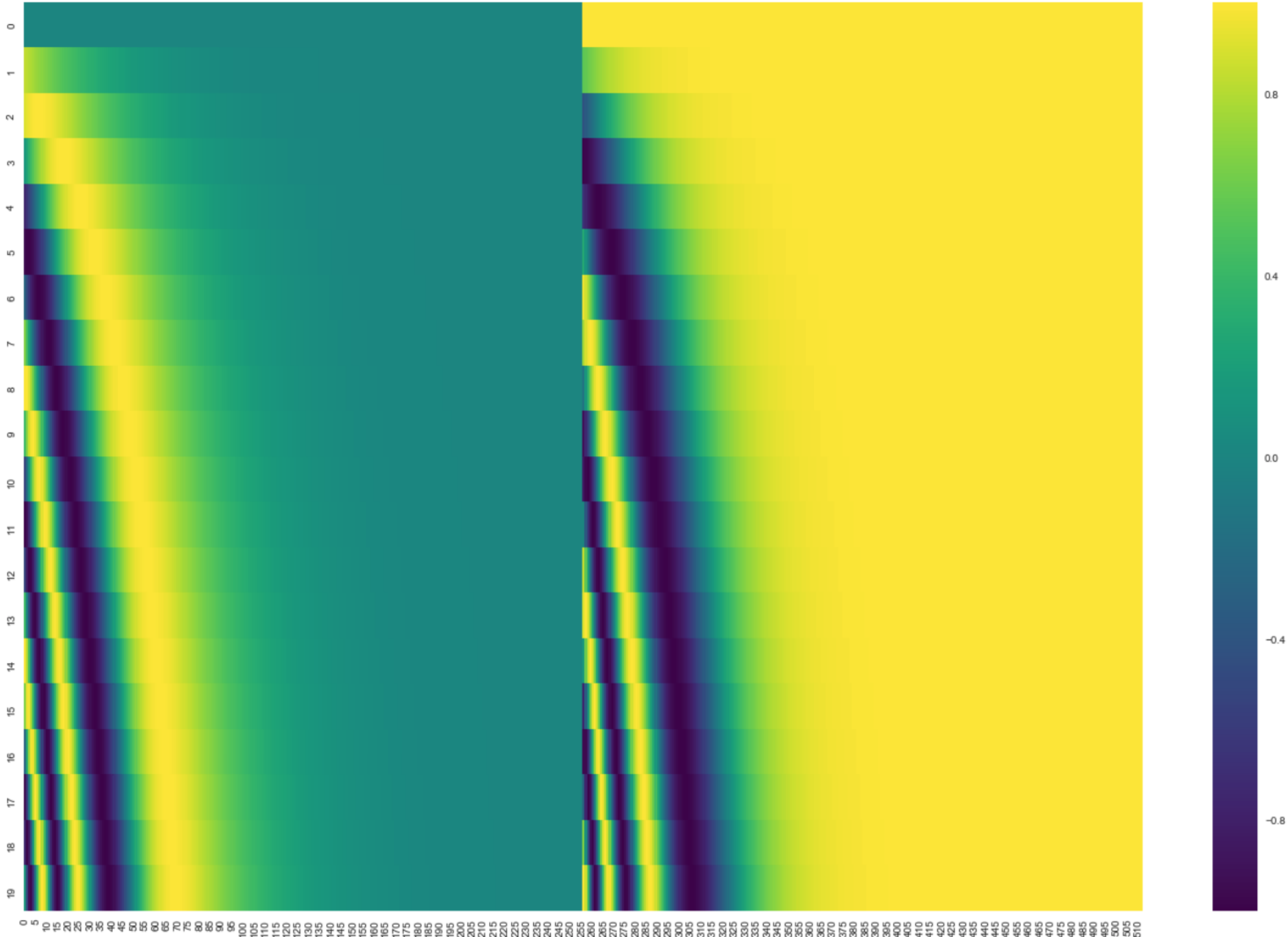
$$X^{(1)} = \mathrm{LayerNorm}(\tilde{X} + F)$$

This is the **output of Layer 1**.

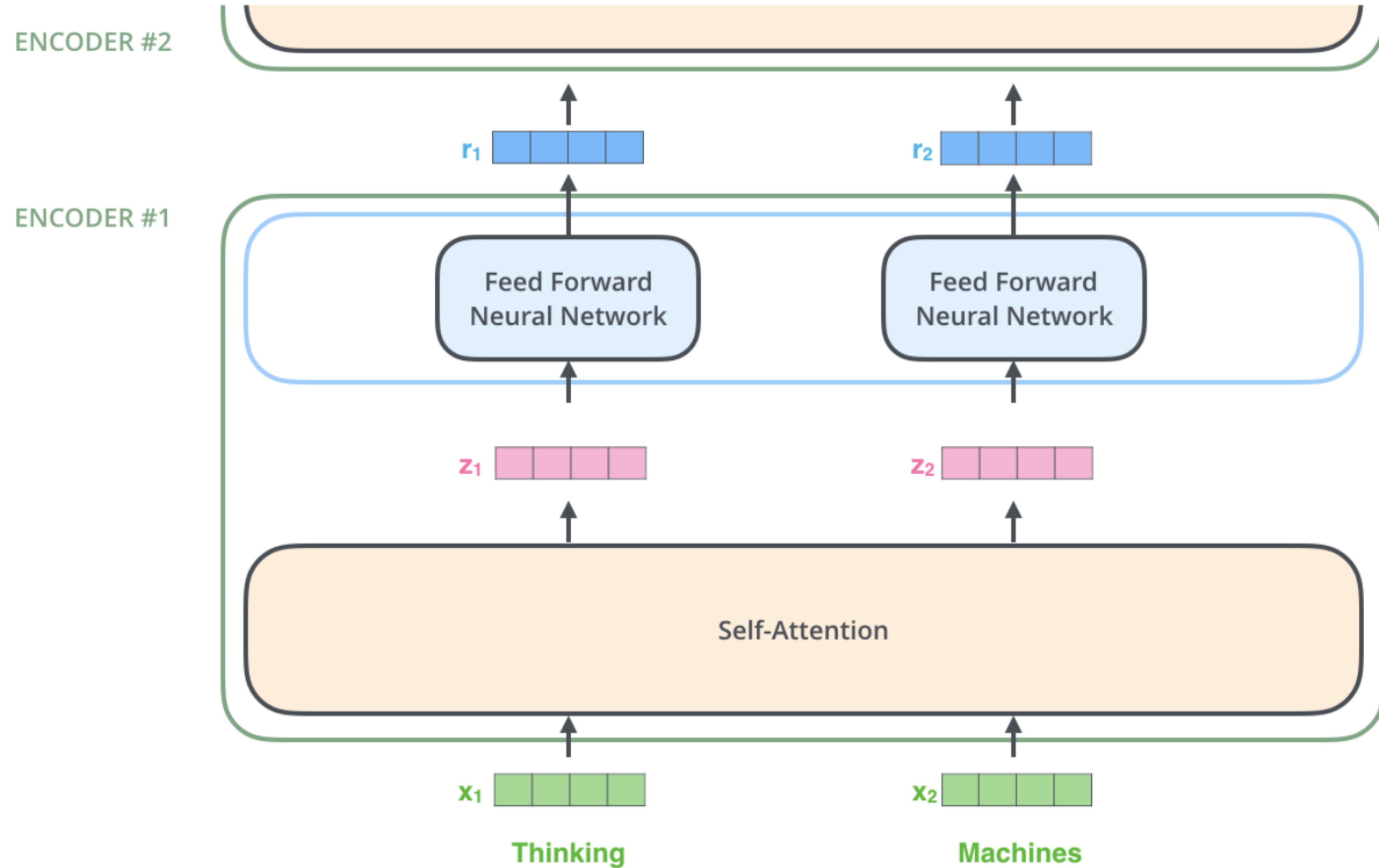
Parsing Encoder: Multi-Head Attention and FFN



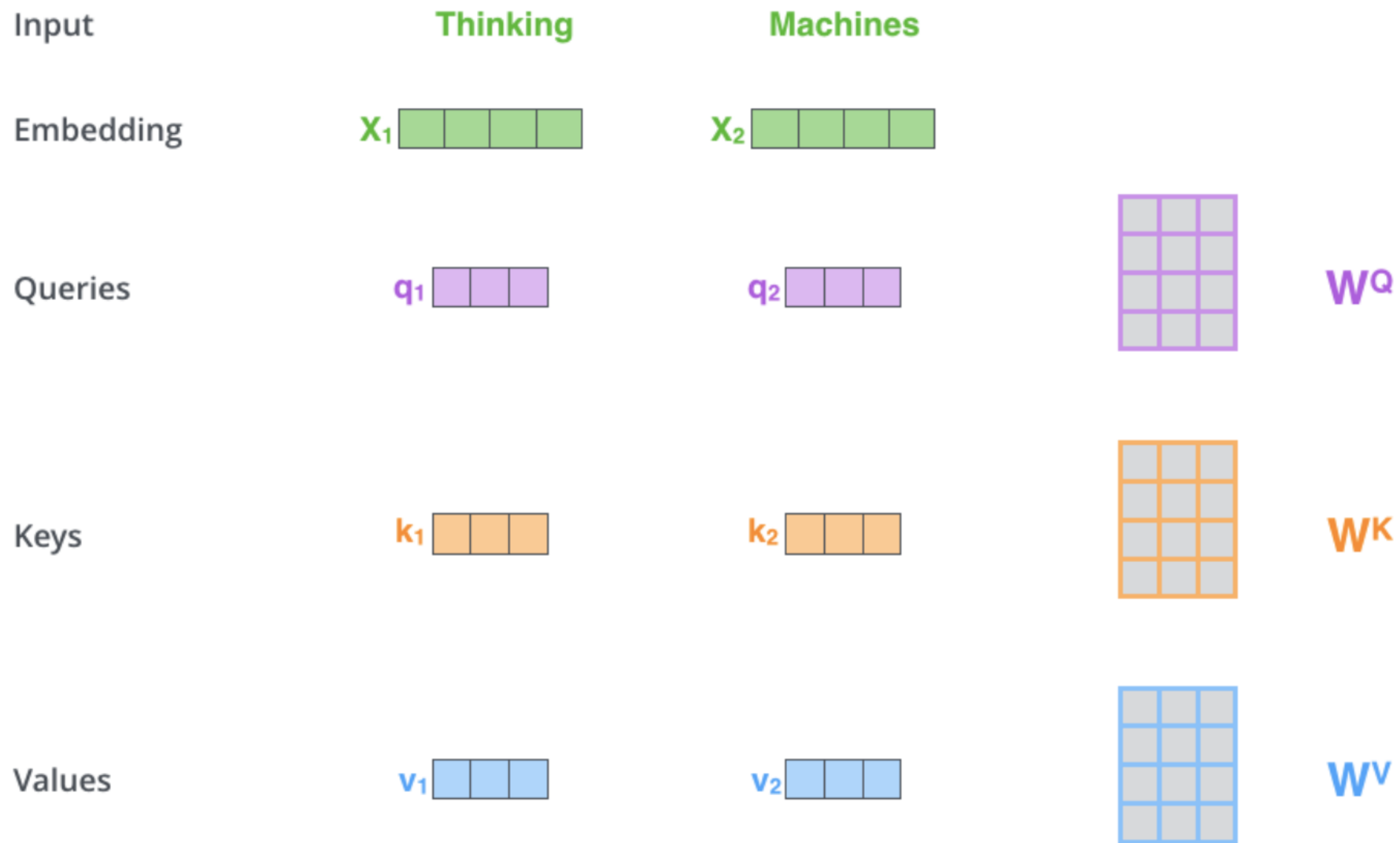
Positional Encoding



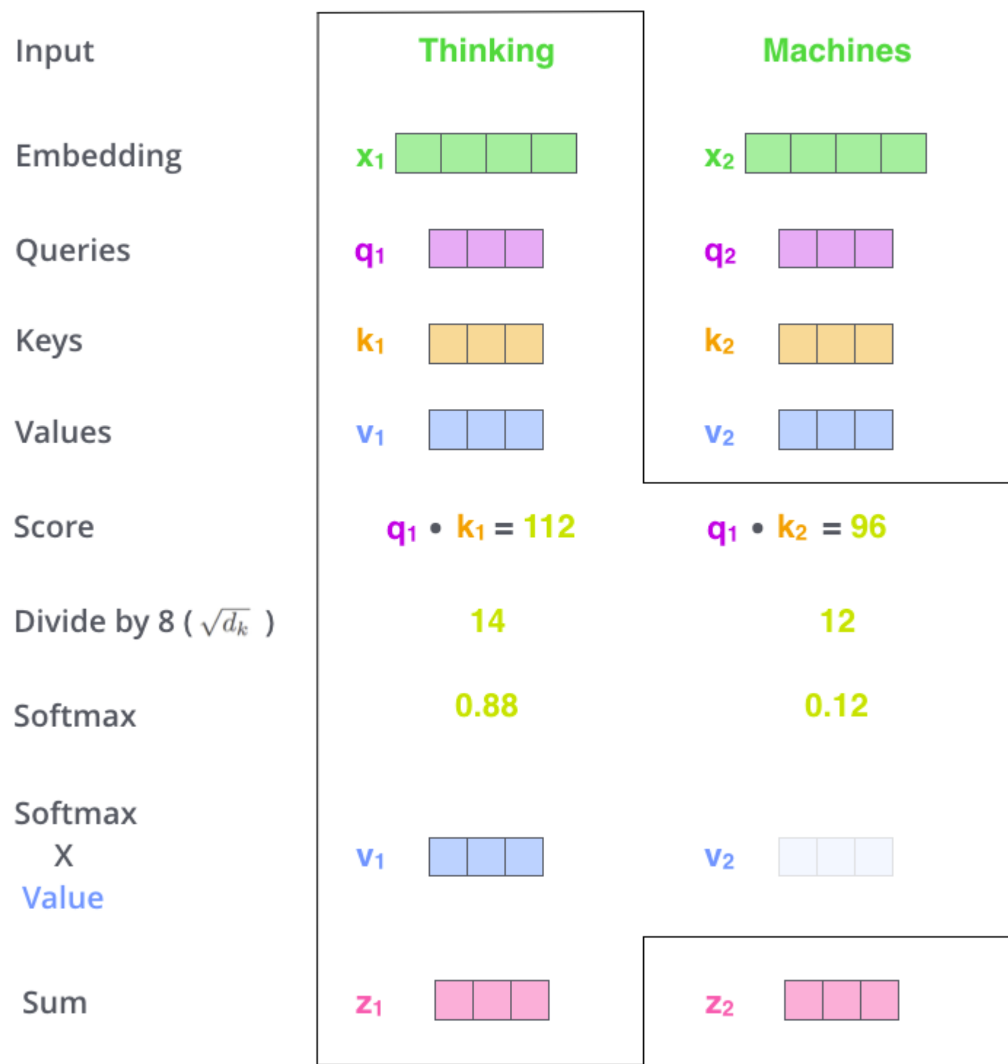
Parsing Encoder: Multi-Head Attention and FFN



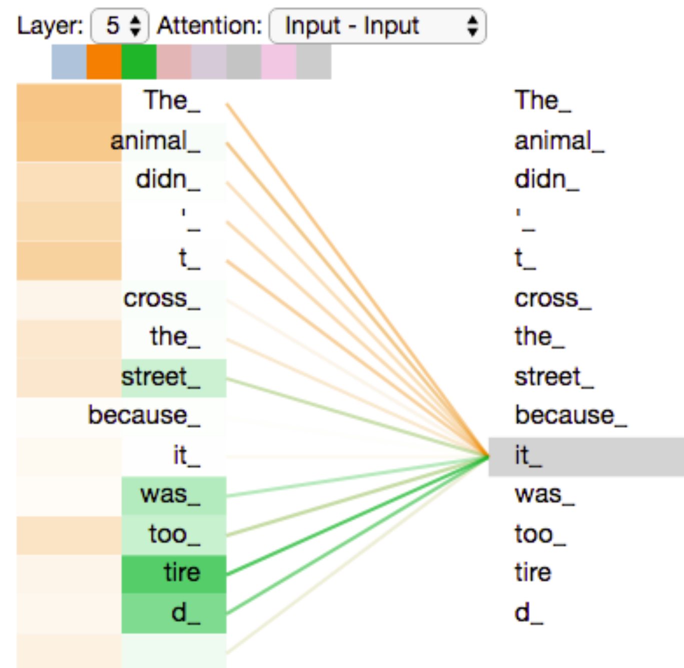
Parsing Encoder: Single Head Attention



Parsing Encoder: Single Head Attention

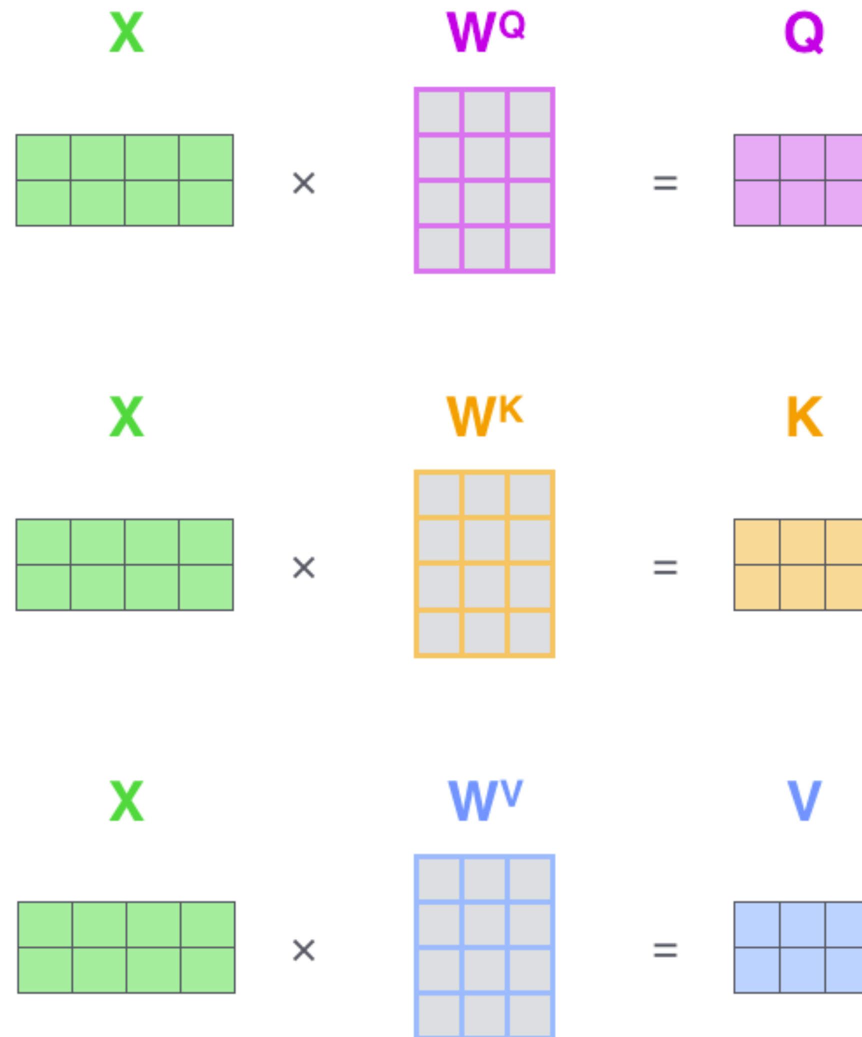


Parsing Encoder: Single Head Attention



As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Parsing Encoder: Single Head Attention



Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

Parsing Encoder: Multi-Head Attention and FFN

1) This is our input sentence*

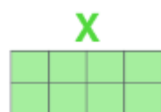
2) We embed each word*

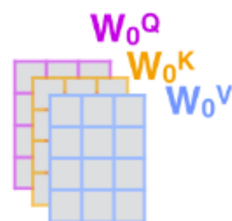
3) Split into 8 heads. We multiply X or R with weight matrices

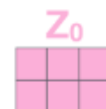
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

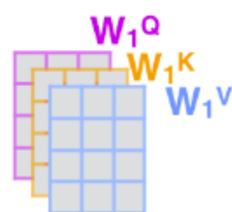
Thinking Machines

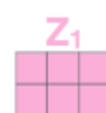


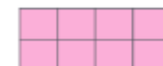




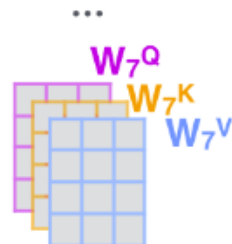
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

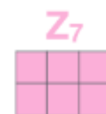




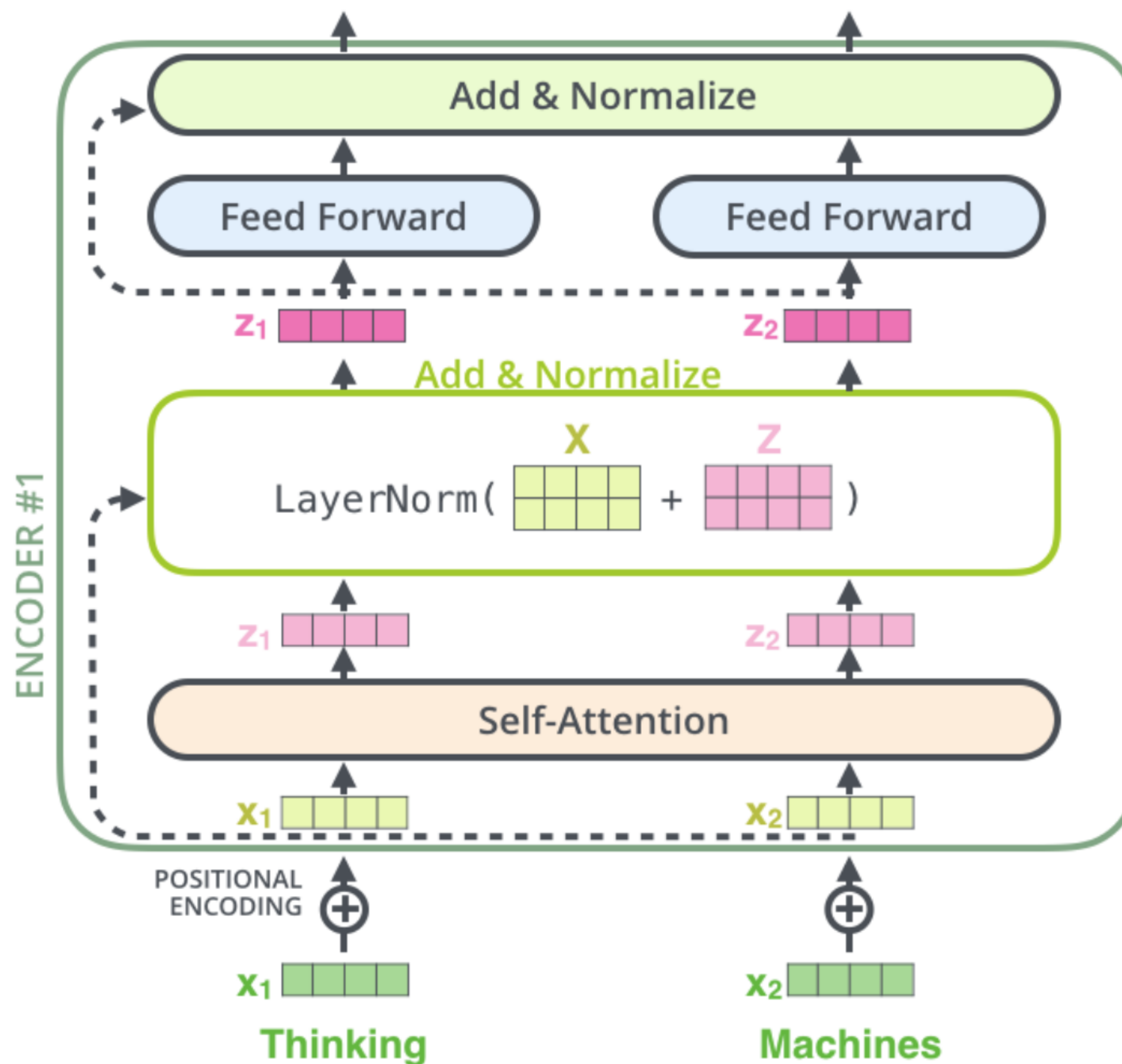


...





Layer Normalization



Self-Attention vs FFN

- Self-Attention: Communication between tokens, better understanding of context for each token
- FFN: Deeper representation of each token. Each token reasons internally, deeper thinking.
- What if there was no self-attention: Reduces to a Deep Neural Network or Recurrent Neural Network
- What if there was no FFN: The deeper learning and deeper representations would be missed

ICE #3: Self-attention Exercise

Let's go through a self-attention python calculation exercise to understand it better. Let $x = [[1, 2, 3, -1], [3, -4, -7, 5]]$ be the input token embeddings. In the first layer of the encoder of the transformer, the weight matrices are given by $W^Q = [[-1, 2, 0], [2, 3, -5], [1, 0, 0], [-3, 1, 2]]$,
$W^K = [[1, 2, 3], [2, 4, 3], [3, 0, 3], [-1, 5, 2]]$,
$W^V = [[-1, -2, 3], [2, -4, 0], [0, 0, 1], [1, 0, -7]]$. Compute the soft-max similar to what we did in the previous walk-through. You can use python matrix multiplication (e.g. numpy) to arrive at the solution. Question is which token (token 1 or token 2) does token 2 place more attention on and what is the attention probability?

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

ICE: Supervised or Un-supervised?

1. Are the above two tasks supervised or un-supervised?

BERT pre-training

Two Tasks

1. **Masked LM Model:** Mask a word in the middle of a sentence and have BERT predict the masked word
2. **Next-sentence prediction:** Predict the next sentence - Use both positive and negative labels. How are these generated?

ICE: Supervised or Un-supervised?

1. Are the above two tasks supervised or un-supervised?

Data set!

English Wikipedia and book corpus documents!

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Cross-Entropy

$L(p, \hat{p}) = -\sum_i [p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i)]$

Loss Function for Masked Language Model (MLM)

Loss Function for MLM mimicks which type of classic ML model?

Cross-Entropy

$L(p, \hat{p}) = -\sum_i \left[p_i \log(\hat{p}_i) + (1 - p_i) \log(1 - \hat{p}_i)\right]$

ICE: What is the loss function for Binary Classification?

Sentence BERT a.k.a sBERT

Uses Siamese Twins architecture

Sentence BERT a.k.a sBERT

Uses Siamese Twins architecture

Advantages of sBERT

More optimized for Sentence Similarity Search.

Sentence BERT - Siamese BERT architecture

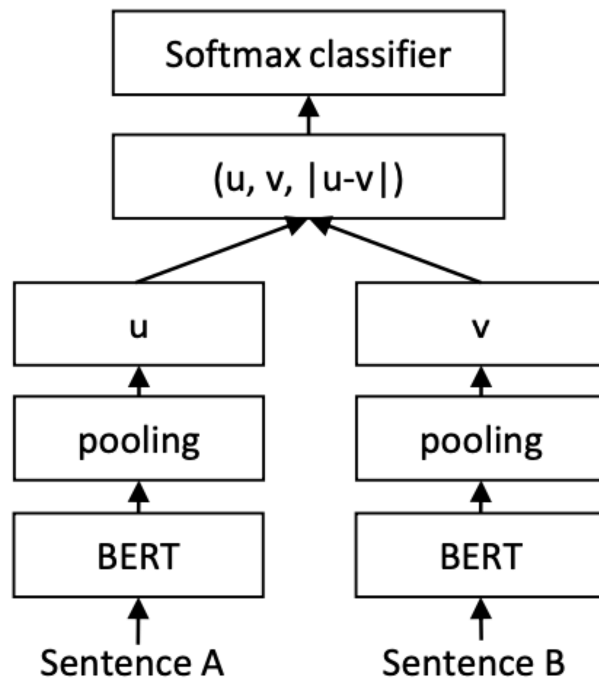


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

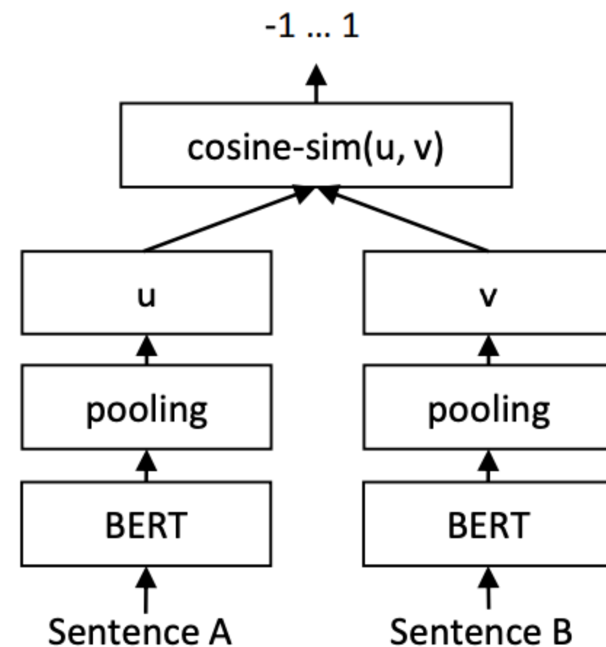


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

Loss Function for SBERT

Pooling Strategy for SBERT

	NLI	STSb
Pooling Strategy		
MEAN	**80.78**	**87.44**
MAX	79.07	69.92
CLS	79.80	86.62
Concatenation		
(u, v)	66.04	-
$(\lvert u-v \rvert)$	69.78	-
$(u * v)$	70.54	-
$(\lvert u-v \rvert, u * v)$	78.37	-
$(u, v, u * v)$	77.44	-
$(u, v, \lvert u-v \rvert)$	**80.78**	-
$(u, v, \lvert u-v \rvert, u * v)$	80.44	-

Table 6: SBERT trained on NLI data with the classification objective function, on the STS benchmark (STSb) with the regression objective function. Configurations are evaluated on the development set of the STSb using cosine-similarity and Spearman's rank correlation. For the concatenation methods, we only report scores with MEAN pooling strategy.

Sentence BERT Cosine Similarity Results

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.

SentEval DataSets

- **MR**: Sentiment prediction for movie reviews snippets on a five start scale (Pang and Lee, 2005).
- **CR**: Sentiment prediction of customer product reviews (Hu and Liu, 2004).
- **SUBJ**: Subjectivity prediction of sentences from movie reviews and plot summaries (Pang and Lee, 2004).
- **MPQA**: Phrase level opinion polarity classification from newswire (Wiebe et al., 2005).
- **SST**: Stanford Sentiment Treebank with binary labels (Socher et al., 2013).
- **TREC**: Fine grained question-type classification from TREC (Li and Roth, 2002).
- **MRPC**: Microsoft Research Paraphrase Corpus from parallel news sources (Dolan et al., 2004).

Sentence BERT on SentEval Results

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	Avg.
Avg. GloVe embeddings	77.25	78.30	91.17	87.85	80.18	83.0	72.87	81.52
Avg. fast-text embeddings	77.96	79.23	91.68	87.81	82.15	83.6	74.49	82.42
Avg. BERT embeddings	78.66	86.25	94.37	88.66	84.40	92.8	69.45	84.94
BERT CLS-vector	78.68	84.85	94.21	88.23	84.13	91.4	71.13	84.66
InferSent - GloVe	81.57	86.54	92.50	**90.38**	84.18	88.2	75.77	85.59
Universal Sentence Encoder	80.09	85.19	93.98	86.70	86.38	**93.2**	70.14	85.10
SBERT-NLI-base	83.64	89.43	94.39	89.86	88.96	89.6	**76.00**	87.41
SBERT-NLI-large	**84.88**	**90.07**	**94.52**	90.33	**90.66**	87.4	75.94	**87.69**

Table 5: Evaluation of SBERT sentence embeddings using the SentEval toolkit. SentEval evaluates sentence embeddings on different sentence classification tasks by training a logistic regression classifier using the sentence embeddings as features. Scores are based on a 10-fold cross-validation.

ICE #4

Let's say we want to automatically convert a **Natural Language Query** to a **SQL** query. E.g. "Which quarter in the past 5 years had the most amount of sales for fashion products" to "SELECT ... FROM ... WHERE ..." What kind of deep learning architecture would support this problem?

1. SBERT
2. LSTM to LSTM sequence model
3. GPT-2
4. Feed Forward Neural Network

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).
3. **Add fine-tuning layers:** Add fine-tuning layers on top of the pre-trained layers. Example, starting with the pooled embeddings, construct one or more dense layers (`Feed-Forward NN style`) to extract finer representations of the input. Add the output layer and its activation (typically softmax for classification tasks).

Fine-Tuning Transformers for down-stream tasks

A methodology for fine-tuning transformers for classification tasks

1. **Pick Base pre-trained Architecture:** Pick a base pre-trained architecture as a starting point for your fine-tuning. Example: `bert-base-uncased` is one such pre-trained model that can be loaded through `Hugging Face Transformers Library`
2. **Extract output from pre-training:** How do you want to use the output from pre-training going into *fine-tuning*? a) Extract embedding from the first token, `CLS` b) Average embeddings of all tokens as a starting point (mean pooling).
3. **Add fine-tuning layers:** Add fine-tuning layers on top of the pre-trained layers. Example, starting with the pooled embeddings, construct one or more dense layers (`Feed-Forward NN style`) to extract finer representations of the input. Add the output layer and its activation (typically softmax for classification tasks).
4. **Set training schedule, hyper-parameters, etc:** Set up optimizer (e.g. ADAM), hyper-parameters, training schedule, etc for training.

ICE #5

BERT Embeddings and Emotion Detection

Let's say you want to do emotion detection by fine-tuning BERT (Encoding Transformer) on a data set. One of the outputs of the *BERT pre-trained model* for a given input is the *last-hidden-state*. This includes an embedding for every token that was passed into BERT.
Let's say you are going to start with the last hidden layer and use that as input for your *fine-tuned* model. This ICE is about the dimensionality of the inputs and outputs. Let's say you have sentences of the kind: "I am looking forward to today! It's going to be a big day" This sentence conveys excitement. There are 13 words in this input and using *word-piece tokenization*, you arrive at 20 sub-tokens as input into the BERT model. The last hidden layer includes an embedding for every single token. Let's say the embedding dimension for a token is 768.

ICE #5 continued

BERT Embeddings and Emotion Detection

There are 13 words in this input and using *word-piece tokenization*, you arrive at 20 sub-tokens as input into the BERT model. The last hidden layer includes an embedding for every single token. Let's say the embedding dimension for a token is 768. For the purpose of emotion detection - You can either use the *CLS* token (Start token) embedding (also called the pooled embedding) or you can take the average of the embeddings of the tokens in the last hidden layer of BERT. a) What's the dimension of the pooled BERT embedding of this particular input example b) What's the dimension of the CLS/Start token embedding in this example? c) what's the total dimension of the last hidden layer?

1. 768, 15630 and 768
2. 768, 768 and 768
3. 15360, 768 and 15630
4. 768, 768 and 15360

ICE #6

Why does pooling of the output need to be done for sequence classification (e.g. emotion detection)?

1. Reduces the dimensionality
2. Averages context from all the tokens
3. Computational concerns for training the fine-tuned model
4. All of the above

Application of SBERT Embeddings to Instacart Recommendations

Instacart Recommendations

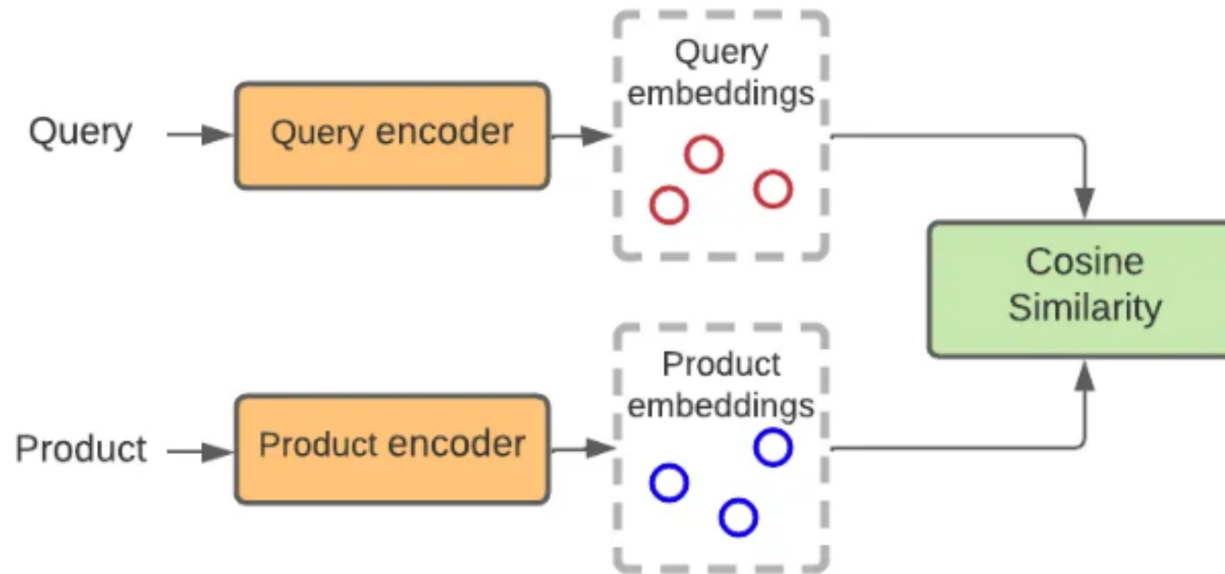
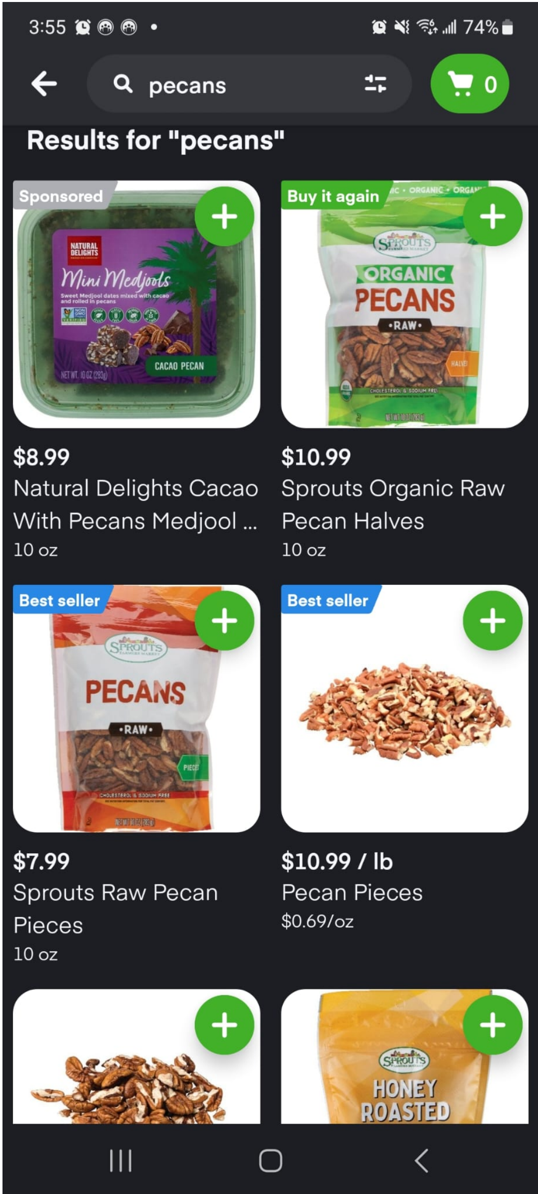


Figure 1. Conceptual diagram of a two-tower model

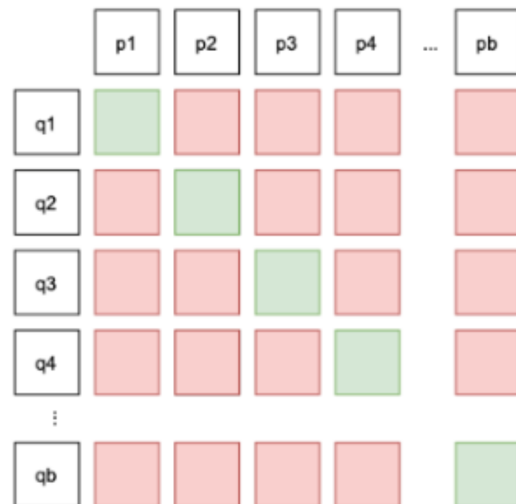
Positive Examples



High-quality Positive Examples

Converted Products for Search Query “Orange”
Navel Oranges
Clementines
Mandarins
...
Bananas
...
Strawberries

Negative Examples



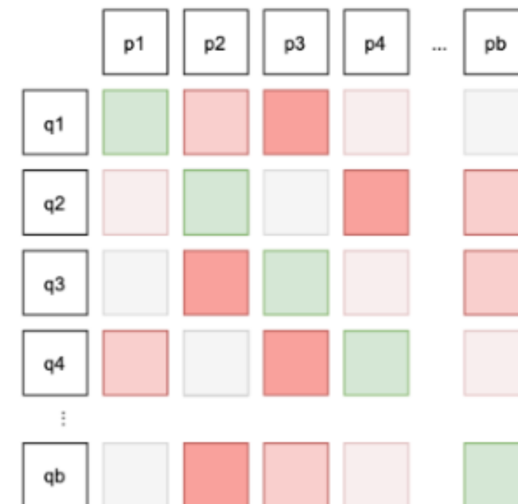


Figure 3. (Left) In the vanilla implementation of in-batch negative, all off-diagonal negative samples are given the same weight. (Right) In our implementation with self-adversarial re-weighting, harder examples are given more weight (darker color), making the task more challenging for the model.

Model Training Architecture

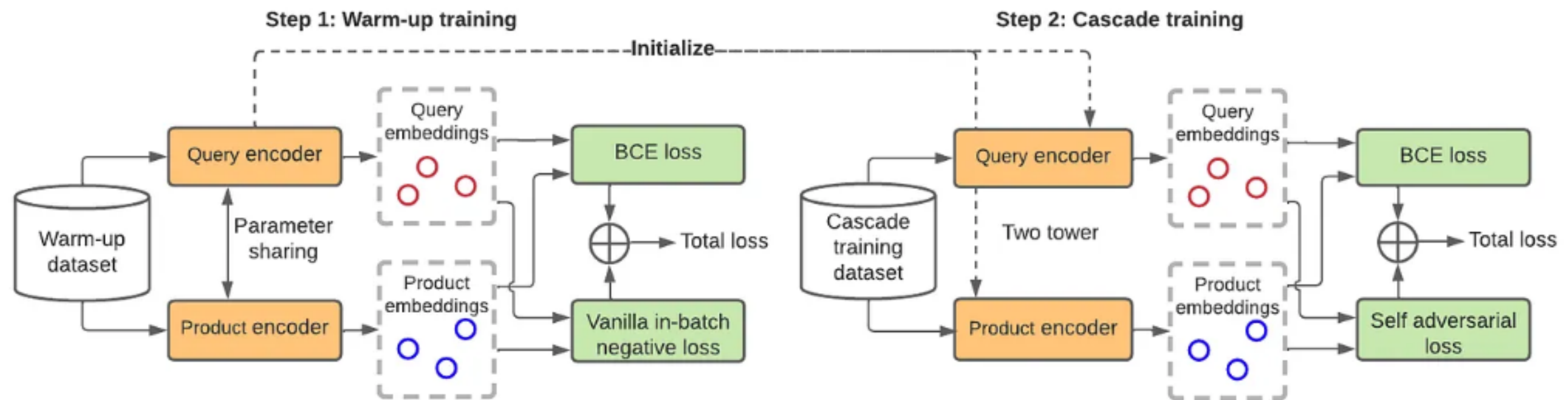


Figure 4. Two-step cascade training for ITEMS.

System Design

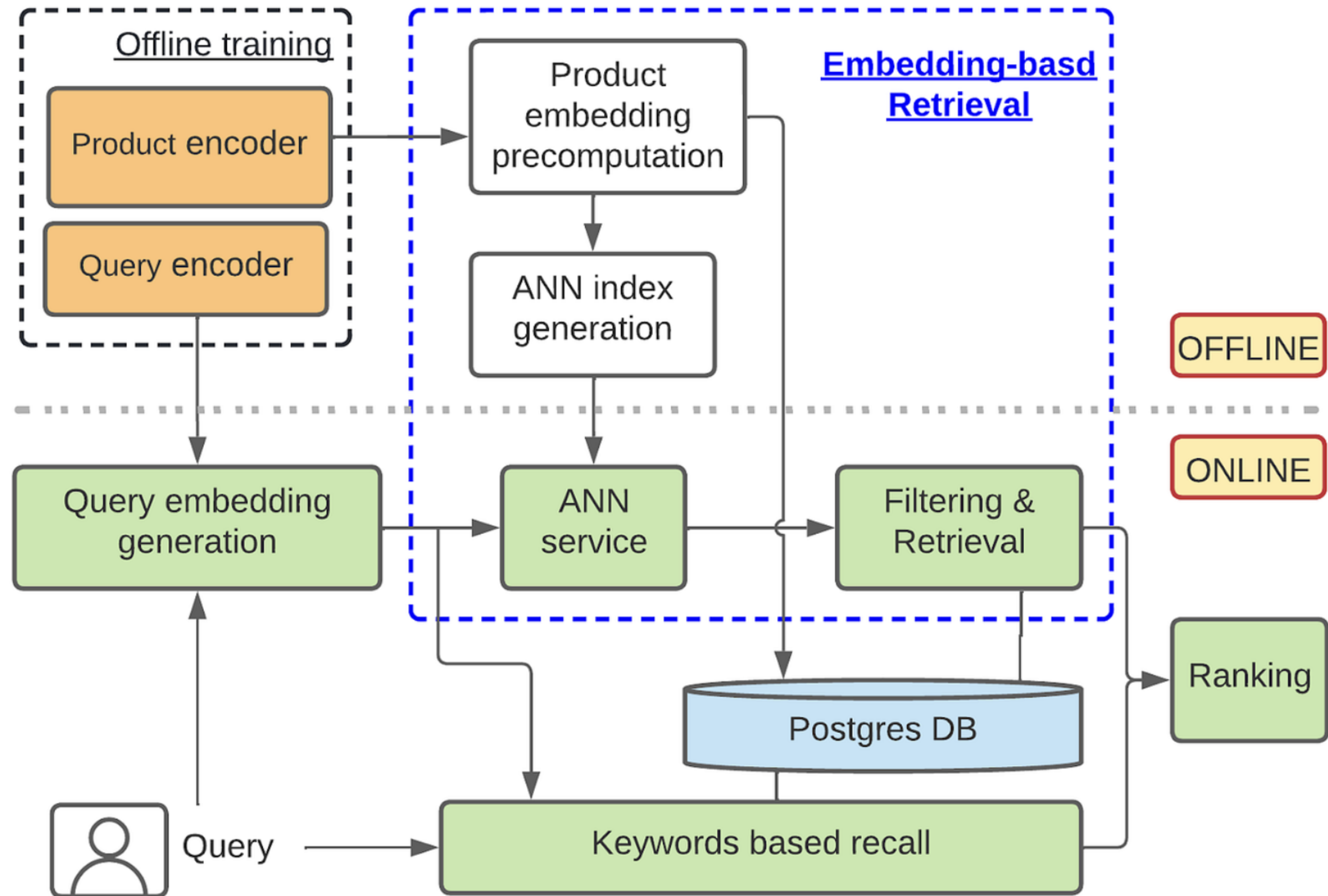


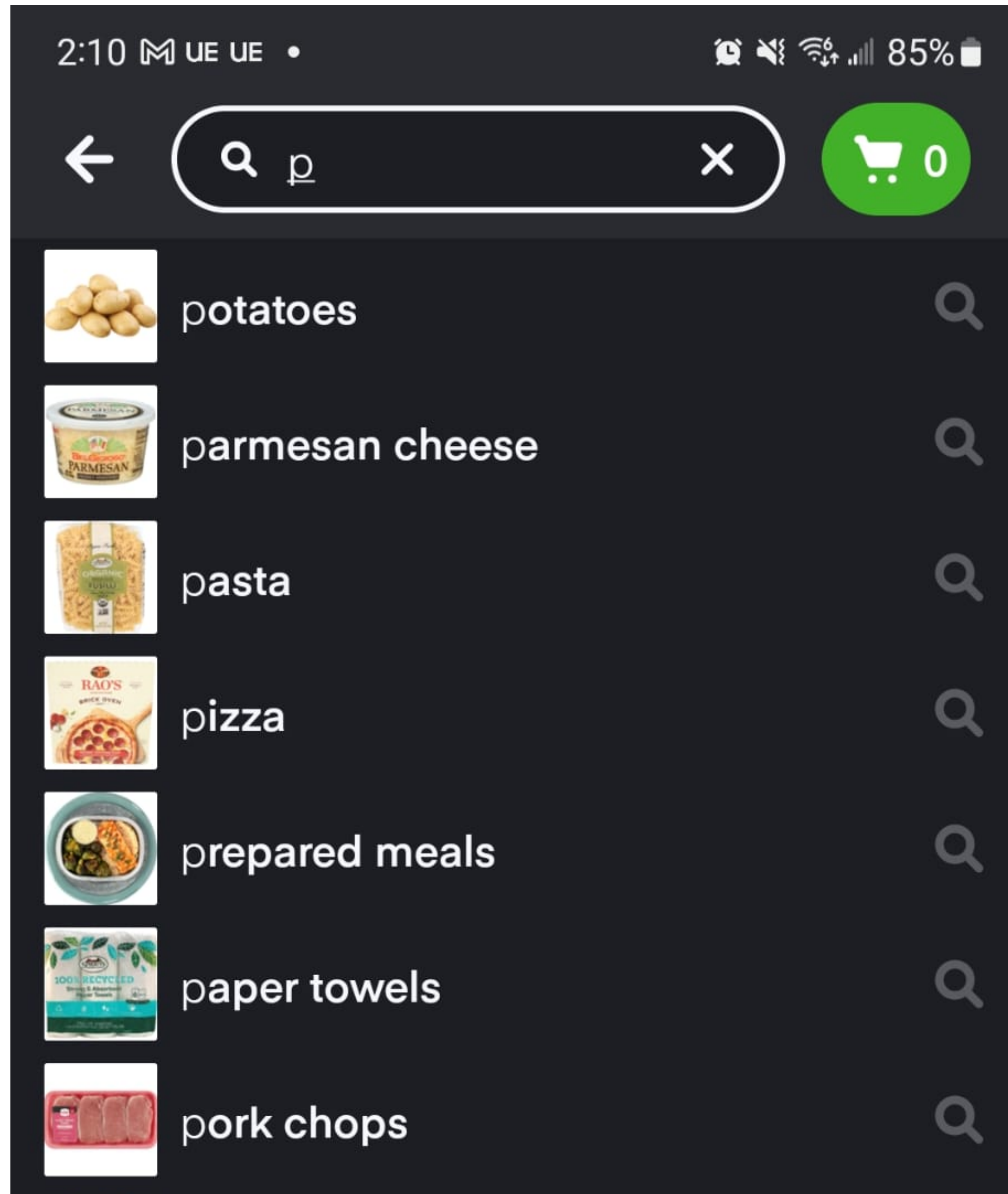
Figure 7. ITEMS system architecture.

Breakouts Time #1

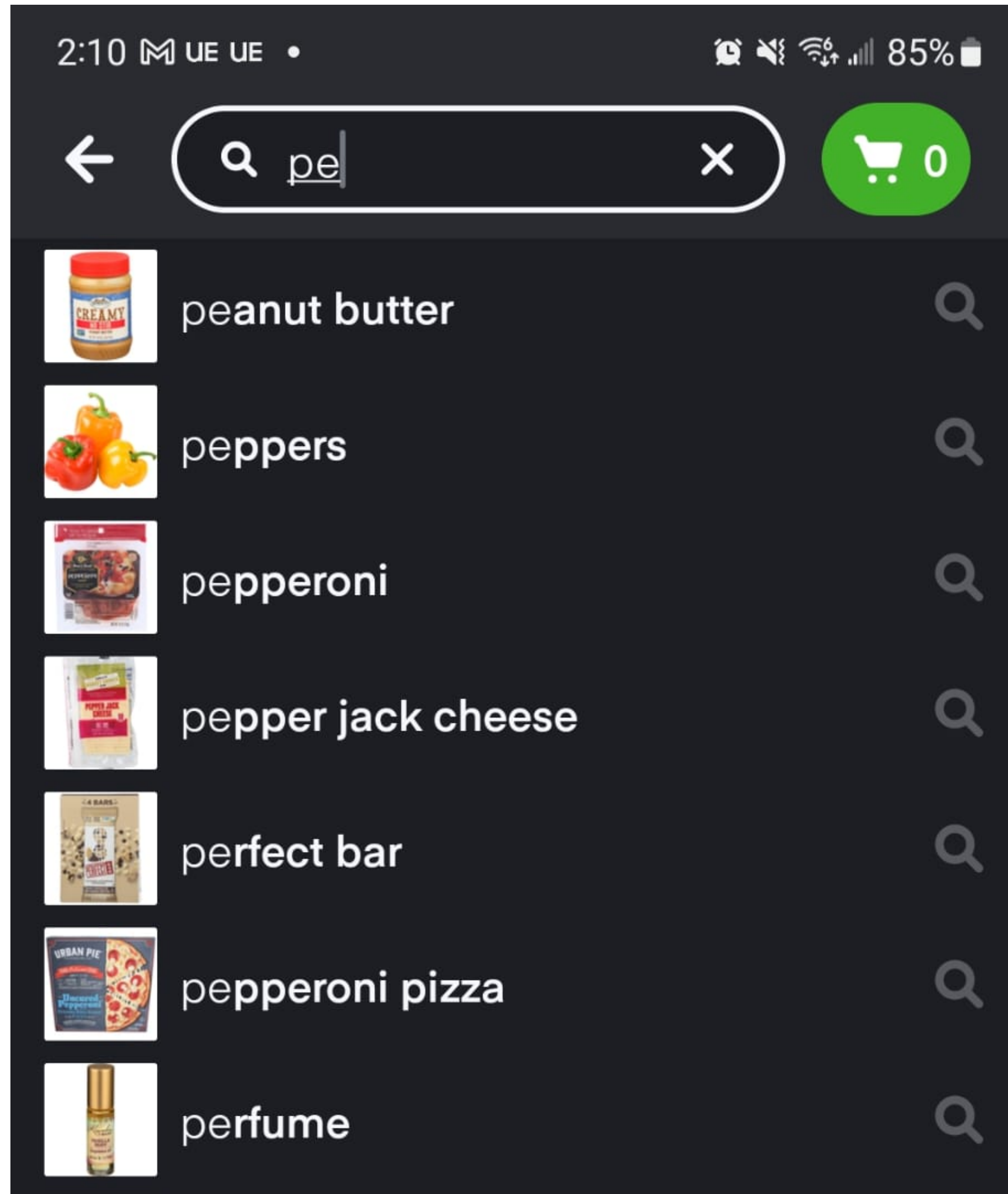
Auto-complete — 5 mins

Let's say you are tasked with building an in-email auto-completion application, which can help complete partial sentences into full sentences through suggestions (auto-complete). How would you use what we have learned so far to model this? What architecture would you use? What would be your data? And what are some pitfalls or painpoints your model should address?

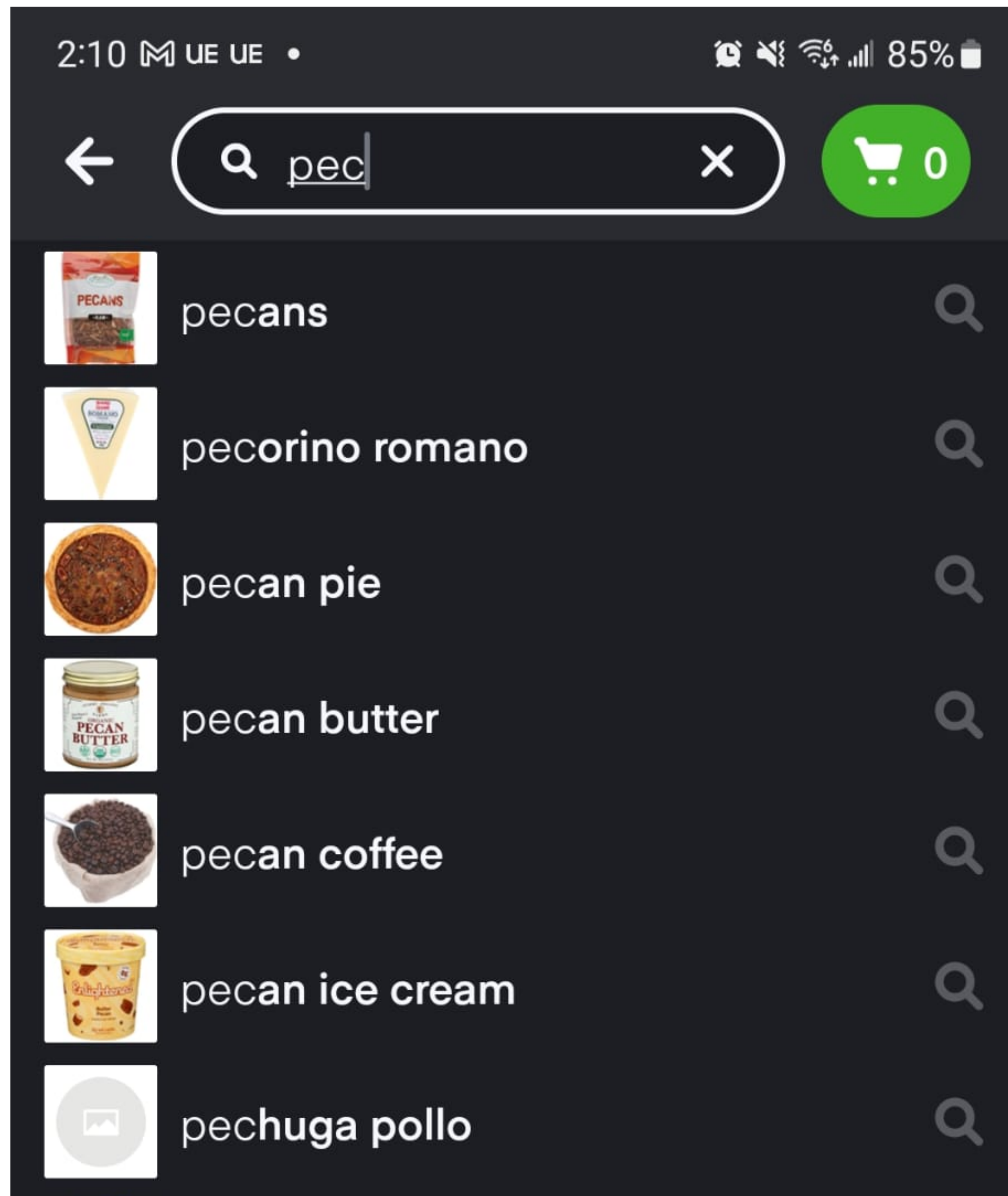
Instacart Auto-Complete and Search Relevance



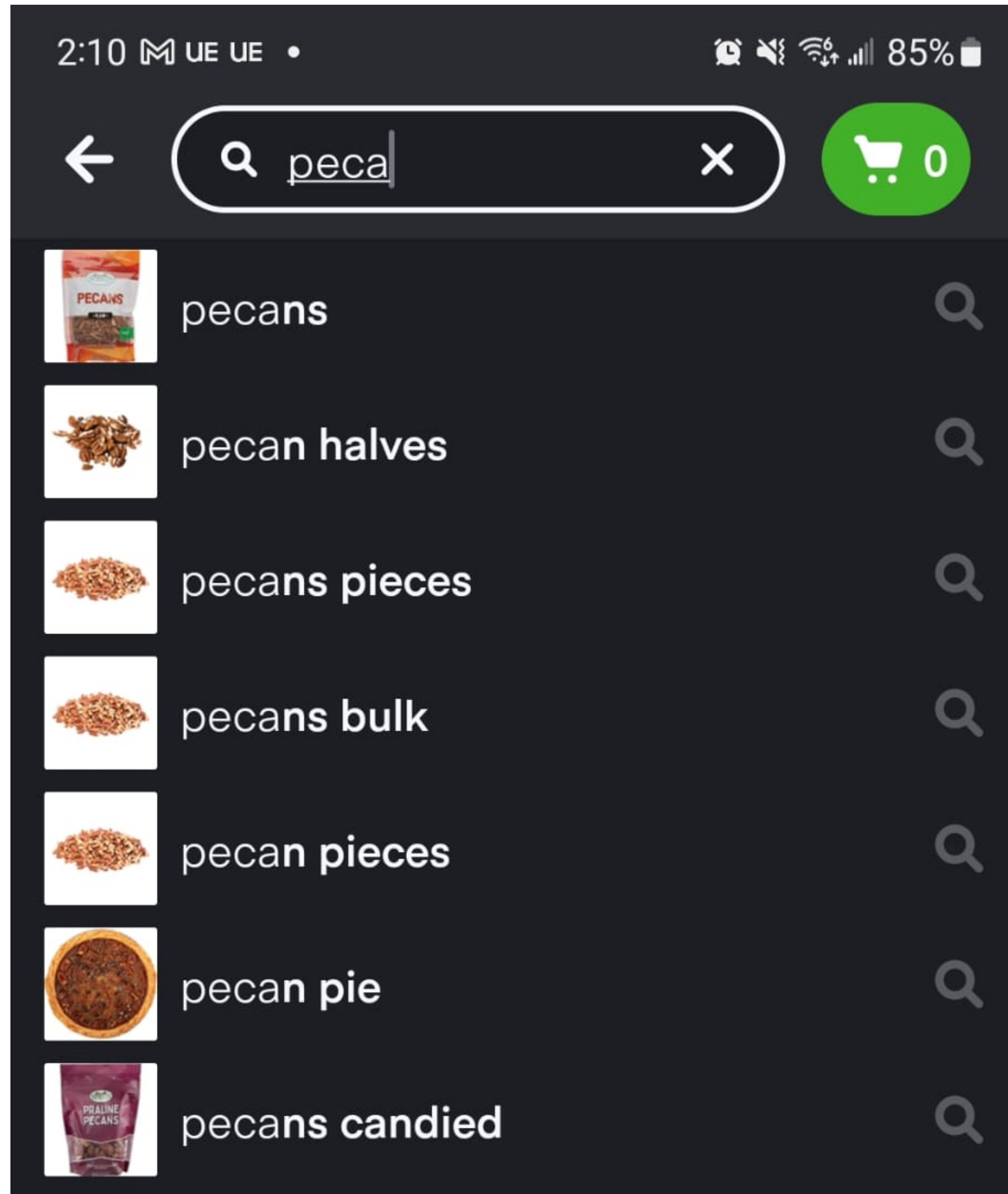
Instacart Auto-Complete



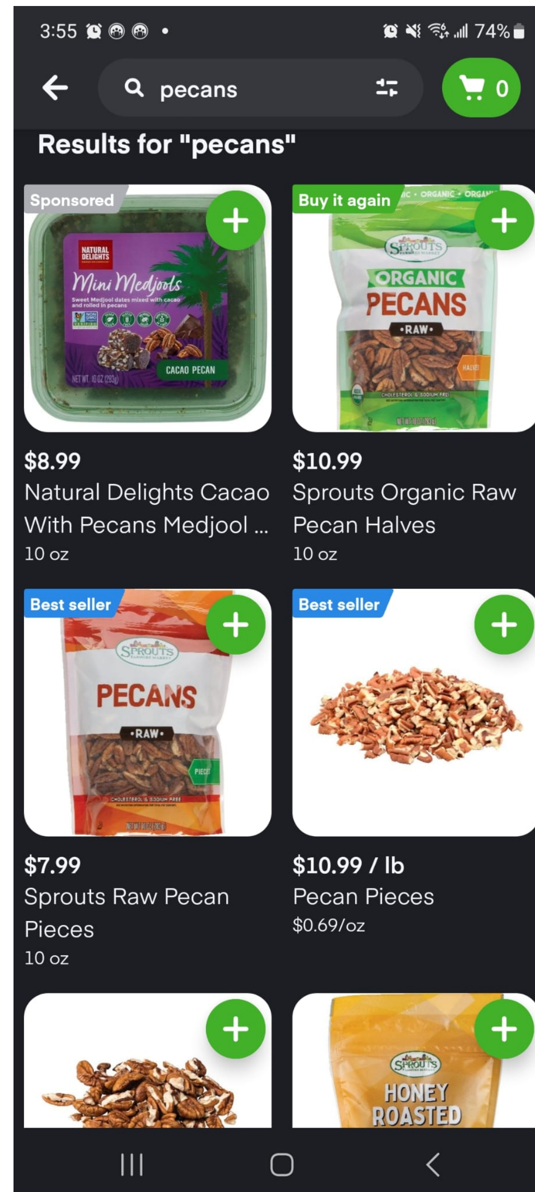
Instacart Auto-Complete



Instacart Auto-Complete



Instacart Auto-Complete and Search Results



Instacart Diversifying Auto-Complete

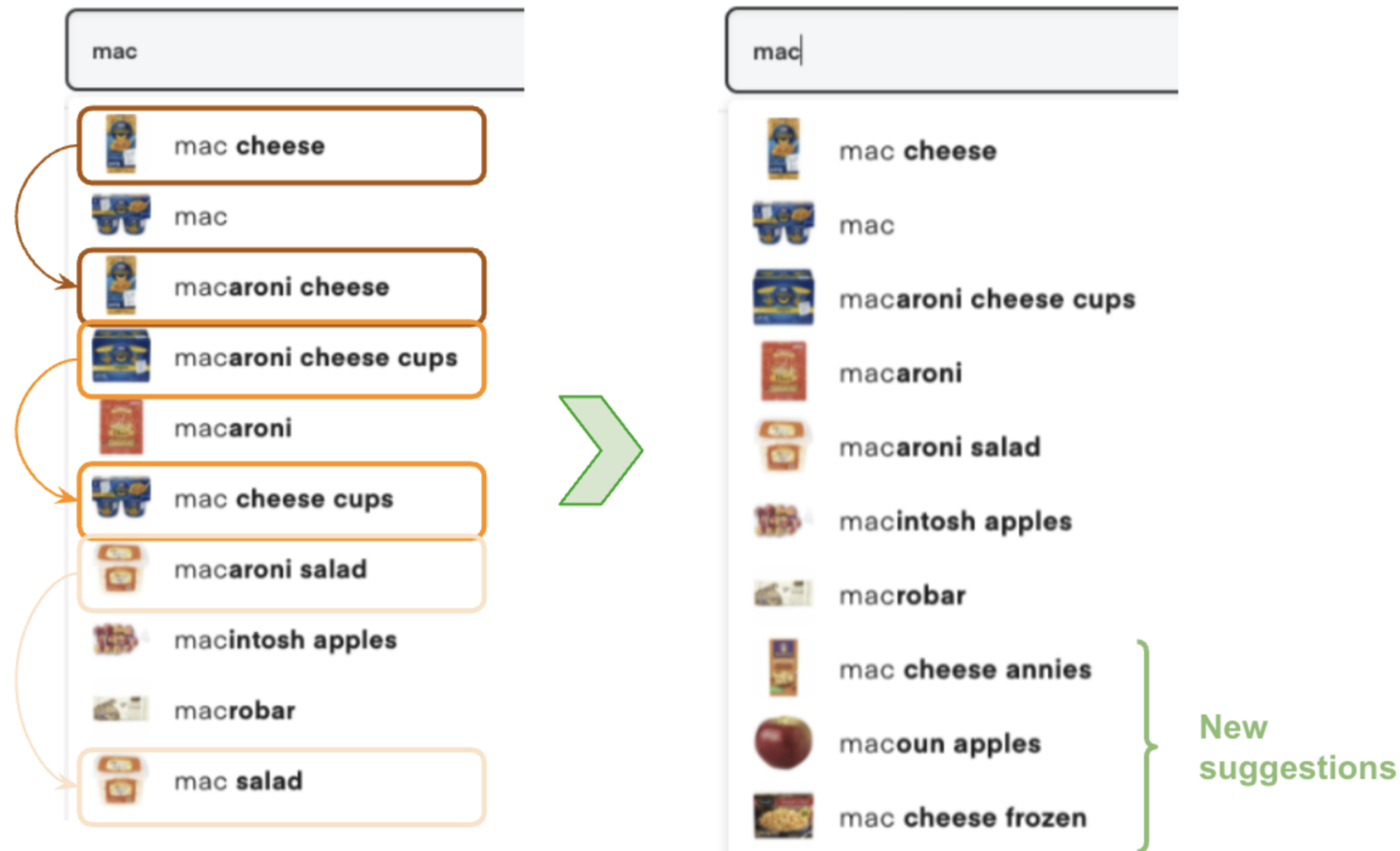


Figure 9. Autocomplete when a customer searches for “mac”, before (left) and after (right) semantic deduplication.