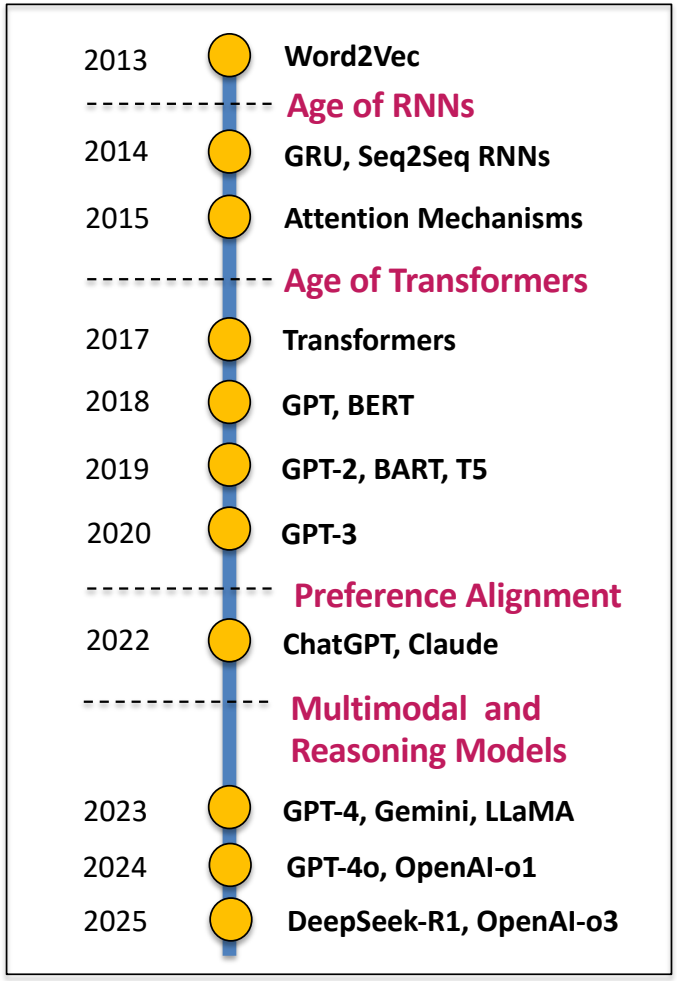
# Tokenization and Word Embeddings

## AI with Deep Learning
## EE4016

**Prof. Lai-Man Po**
Department of Electrical Engineering
City University of Hong Kong

# LLMs: From Word2Vec to DeepSeek-R1 (2012-2025)

1. Tokenization and Word2Vec: BPE, CBOW, Skip-Gram
2. RNNs, LSTM, GRU, Seq2Seq RNNs and Attention Mechanisms
3. Transformers with Self-Attention
4. Lager Language Models (LLMs): BERT, GPT, BART, T5
5. Preference Alignment by SFT and RLHF: ChatGPT, Claude, LLaMA
6. Multimodal Models: GPT-4, GPT-4o, Gemine, LLaVA
7. Reasoning Models: OpenAI-o1, DeepSeek-R1

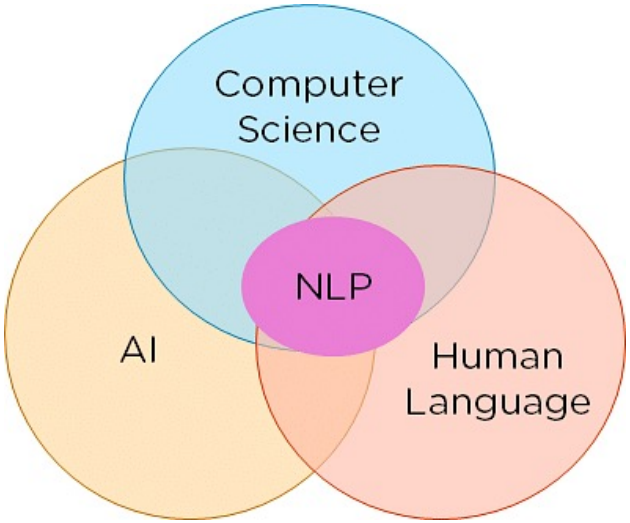


# Outline

- **Introduction of Natural Language Processing (NLP)**
- **Numeric Representations of Text**
  - **Tokenization:** Splitting text into individual tokens (sub-words)
    - Binary Paired Encoding (BPE), WordPiece, SentencePiece
  - **Word Embeddings:** Convert tokens into numerical vectors
    - One-Hot-Encoding, N-Gram, Neural Probabilistic Language Model (NPLM), Word2Vec (CBoW and Skip-Gram), Glovec, and FastText

# What is Natural Language Processing (NLP)?

- **NLP** is a subfield of AI that **enables computers to understand, interpret, and generate human language through algorithms and models.**
- In general, NLP combines computer science, AI, and linguistics to bridge the gap between human communication and computer understanding.
- **Common NLP Tasks and Applications:**
  - Translation
  - Question Answering
  - Summarization
  - Document Classification
  - Text Generation
  - Assisted Writing



# Natural Language Processing Tasks

- **Natural Language Understanding (NLU)**
  - **Sentiment Analysis**: This is a classification task. It analyzes and interprets text to determine their emotional inclination. The result is usually a binary (positive and negative) or triple (positive, neutral or negative)
  - **Text Classification**: This is related to sentiment analysis though encompasses more.
  - **Natural Language Inference (NLI)**: Determination of whether a given "hypothesis" logically follows from a given "premise". (Entailment, Contradiction, and Neutral)
  - **Semantic Understanding**: Interpretation and comprehension of words, phrases, sentences and the relationships between them such as Named Entity Recognition (NER).
- **Natural Language Generation (NLG)**
  - **Summarization**: Ability to create a concise abstract for a given sentence or paragraph.
  - **Question Answering**: Creating answers to specific questions
  - **Chatbot**: A conversational AI system that engages with users in a natural, human-like way.
- **Reasoning and Math**
  - To advance NLU and NLG, models must develop reasoning and math skills to interpret complex problems, infer implicit information, and solve multi-step tasks that go beyond surface-level text understanding.

# Challenges in NLP

NLP algorithms must overcome **two primary linguistic challenges** that confuse logic-based systems:

## 1. Synonymy Challenges (同义字)

- **Multiple words, similar meanings.**
- Machines struggle to capture that these represent the same entity. (Poor Recall)


## 2. Polysemy Challenges (一字多义)

- **Single word, multiple meanings.**
- Machines struggle to disambiguate meaning without context. (Poor Precision)


**Financial Bank vs. River Bank.**

# Text Tokenization and Word Embeddings

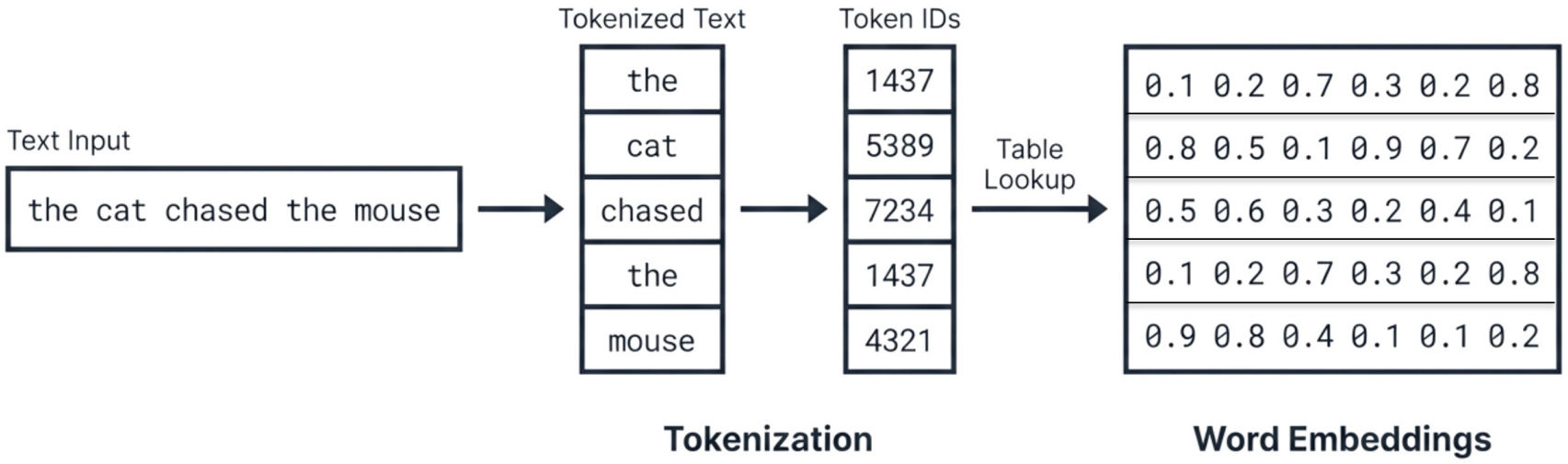
## Unveiling Semantic and Syntactic Properties

https://medium.com/@lmpo/tokenization-and-word-embeddings-the-building-blocks-of-advanced-nlp-c203b78bfd07

# Text Representation in NLP

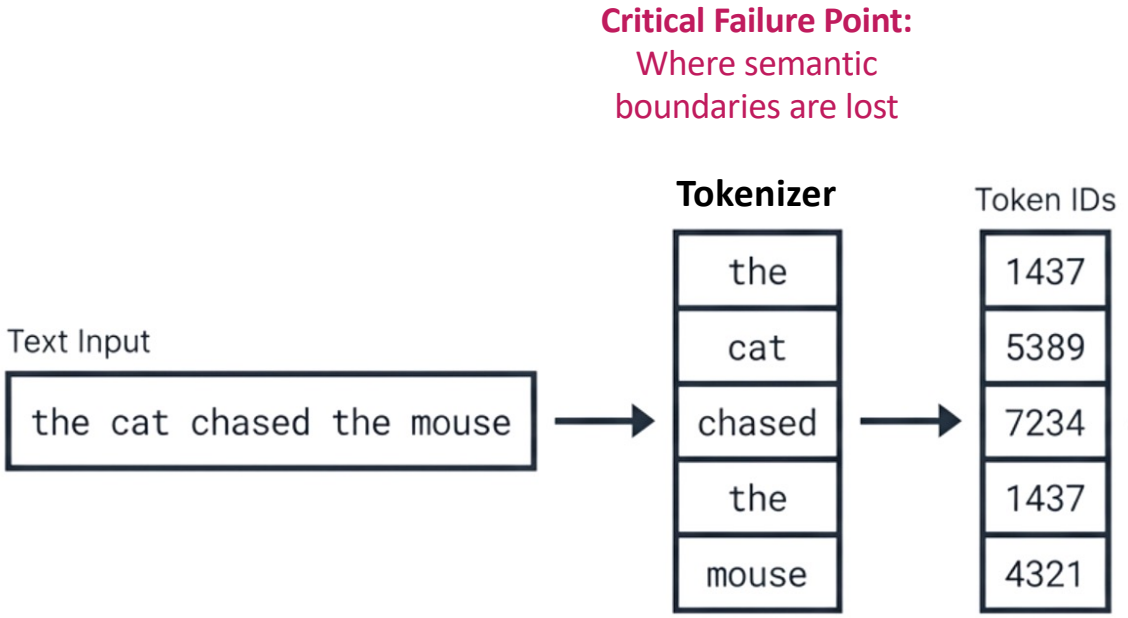
## Neural Networks Do Not Read Text. They Process Vectors.

1. **Tokenization**: Splits text into **tokens** (character, word, subword) for standardization and it is the translation layer between string and integer (Token ID).
2. **Word Embeddings**: Dense vectors capturing semantic relationships.



# The Mandatory Translation Layer

- Tokenization is not merely a preprocessing step; it is the structural interface between the continuous landscape of human language and the discrete numerical domain of machine.
- If the map (tokens) does not match the territory (linguistic meaning), the model builds its reality on a fractured foundation.



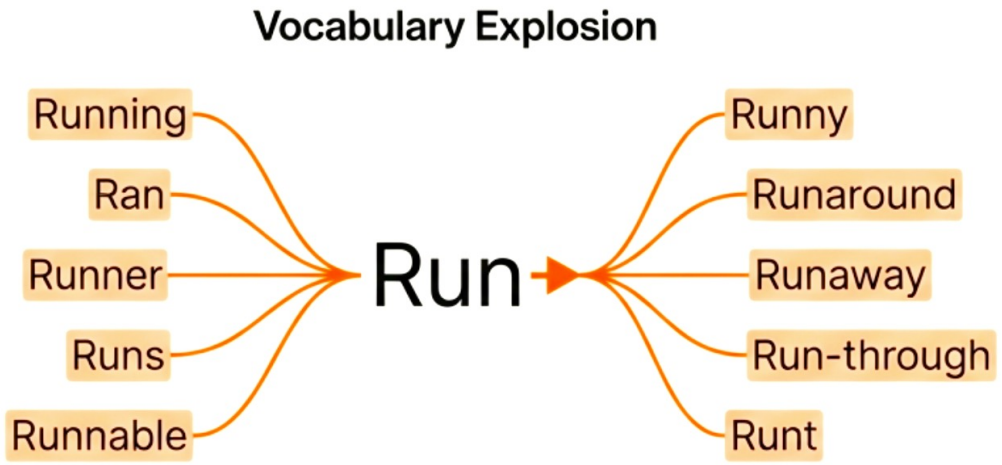
# Three levels of Tokenization

- Tokenization strategies vary based on the **granularity of the tokens** generated.
- The three primary types of tokenization are
  - **Character-level :** “hello” → [”h”, ”e”, “l”, “l", "o"].
  - **Word-level :** “I love coding” → ["I", "love", "coding"].
  - **Subword-level** : “unbelievable” → [”un", ”believ", ”able"].
- Each approach has its strengths and weaknesses, and the choice depends on the specific requirements of the NLP task.

# Approach 1: Word-Level Tokenization

## The Semantic Trap

- Input: "geeksforgeeks is a fantastic resource"
- Output: [ 'geeksforgeeks', 'is', 'a', 'fantastic', 'resource']



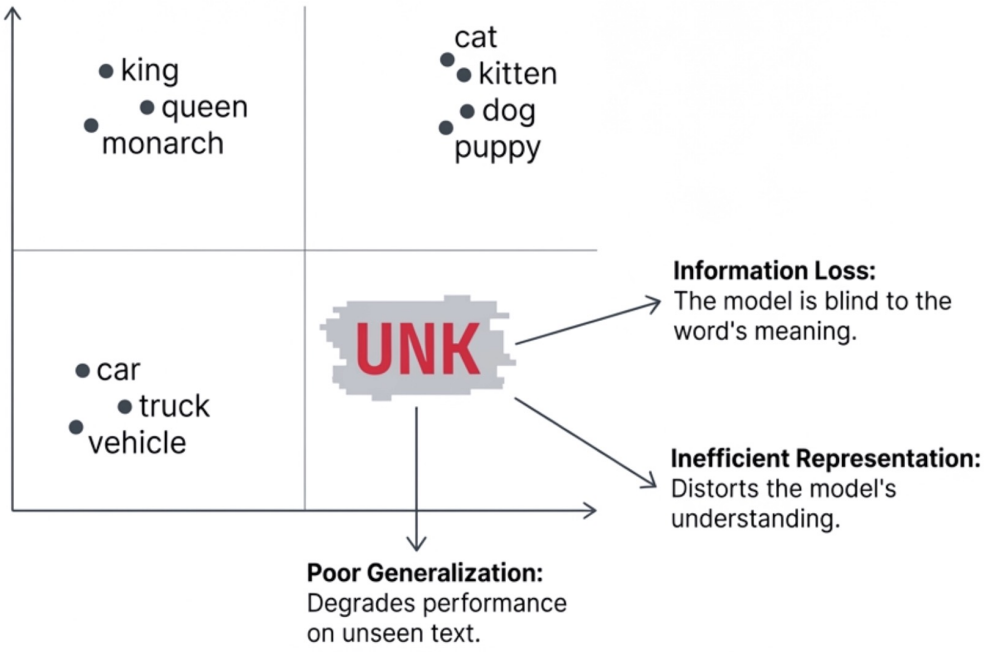

**Fatal Flaw:** 1 Word = 1 Token

- To cover English, a model needs millions of unique tokens.
- Any word not in the training dictionary (like "uninstagrammable" or typos) becomes [UNK]-the Unknown Token.

# The Out-Of-Vocabulary (OOV) Problem

- **Vocabulary Explosion**: To cover a language using word-level tokenization, the vocabulary size can grow to millions of tokens, creating massive memory overhead.
- **Fixed Limit**: Models typically cap vocabularies (e.g., 30,000–50,000 words) for efficiency.

- **The OOV Problem**:
  - Any word **not** in the fixed vocabulary is replaced by a generic **Unknown Token** (e.g., [**UNK**]).
  - **Consequence**: The model loses all semantic information for that word. "Unseen" effectively becomes "meaningless."



# Approach 2: Character-Level Tokenization

## The Coverage Trap

- Splits text into individual characters.
  - **Example:** “Machine learning is powerful.” →
  - [“M”, “a”, “c”, “h”, “i”, “n”, “e”, “ “, “l”, “e”, “a”, “r”, “n”, “i”, “n”, “g”, “ “, “i”, “s”, “ “, “p”, “o”, “w”, “e”, “r”, “f”, “u”, “l”, “.”]
- **Pros:**
  - Handles 'Out of Vocabulary (OOV) perfectly; small vocabulary size.
- **Cons:**
  - **Sequence Dilution.** The model struggles to learn that 'a' + 'p' + 'p' + 'l' + 'e' equals a fruit. Sequences become 5x-8x longer, drastically increasing computational cost.

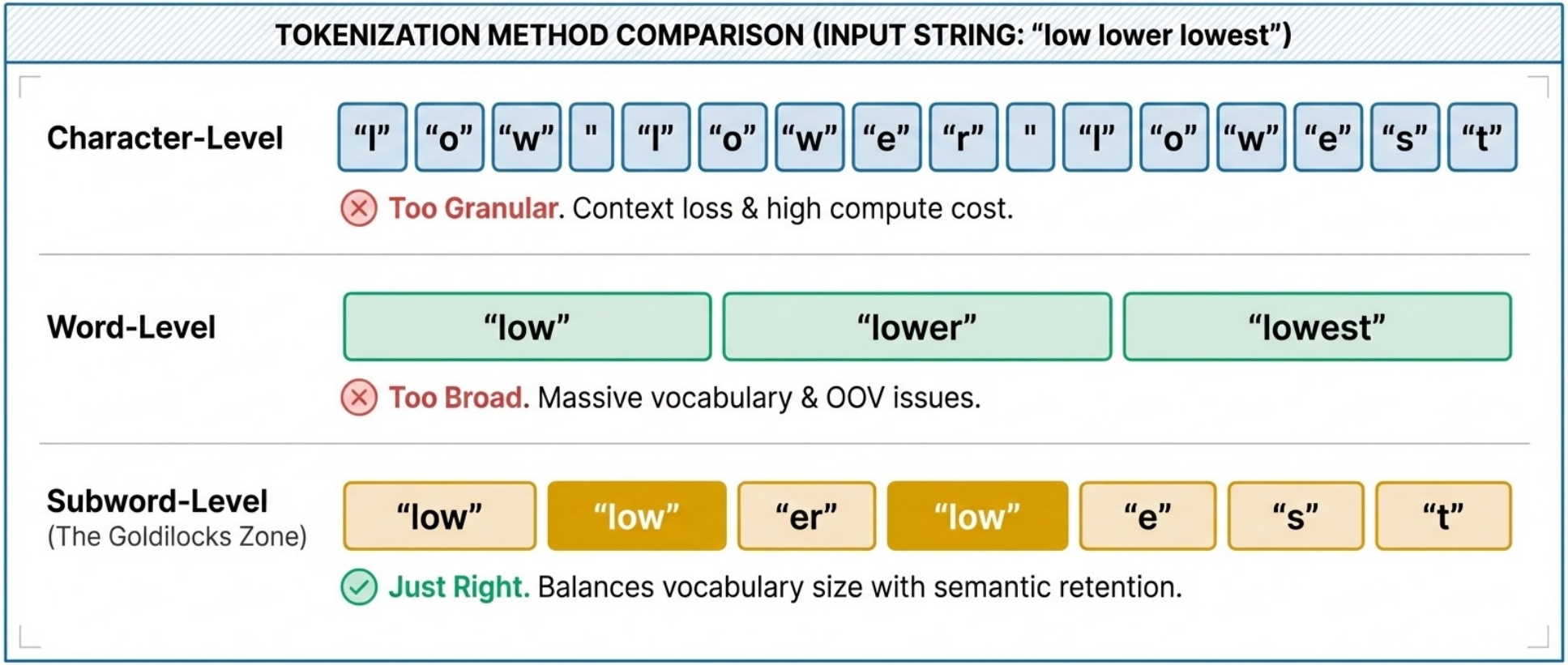
# Approach 3: Subword Tokenization

## The Industry Standard (BERT, GPT, Gemini)

- **Linguistic Insight:** Rare words are usually compounds of common parts (morphemes, roots, affixes).
  - Sennrich et al. found that in 100 rare German tokens, 56 were compounds and 21 were names.
  - Subwords unlock the meaning of these without needing the full word in the dictionary.
- **How It Works:**
  - Frequent words are kept intact as single tokens (e.g., "the", "data") for efficiency.
  - Rare or complex words are decomposed into common subcomponents (e.g., "unbelievable" → "un", "##believ", "##able").
    - Here, "##ed" is a suffix token indicating it's a continuation of the previous subword (not a standalone word).
- **Fixed Vocabulary Size + Near-Zero OOV**

# Finding the Balance Between Granularity and Context

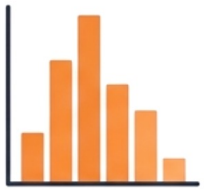
## Why Subword Tokenization is the industry standard.



# The Three Titans of Subword Tokenization.

## BPE

Byte Pair Encoding



The 'Frequency' Master.
Used by `GPT / OpenAI`.

## WordPiece

WordPiece

The 'Probability' Master.
Used by `BERT / Google`.

## SentencePiece

SentencePiece

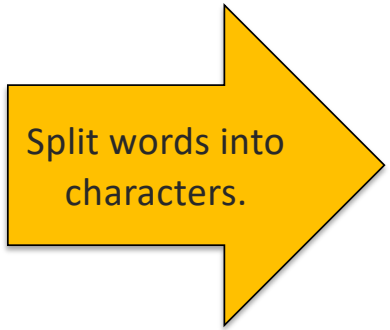
The 'Universal' Master.
Used by `T5 / Multilingual Models`.

# Byte-Pair Encoding (BPE, 1994)

## Building vocabulary through frequency.

- **BPE** is a **text segmentation algorithm** which is also useful for building subword vocabularies by merging frequent pair of bytes.
  - For example: We have a dataset and extracted words from that dataset, and we come up with below count.

| Word | Frequency |
|---|---|
| Most | 3 |
| West | 3 |
| stem | 2 |
| strike | 2 |
| newest | 1 |




| Character Sequence | Frequency |
|---|---|
| M o s t | 3 |
| W e s t | 3 |
| s t e m | 2 |
| s t r l k e | 2 |
| n e w e s t | 1 |

https://medium.com/nerd-for-tech/nlp-tokenization-2fdec7536d17

# Initial Vocabulary List with all Character

- We will generate **a vocabulary size of 13** and it means we will create 13 token out of the character sequence.
- **First of all, we will take all the tokens from character sequence.**

| Character Sequence | Frequency | Vocabulary |
|---|---|---|
| M o s t | 3 | m,o,s,t,e,w,r,l,k,n |
| W e s t | 3 | |
| s t e m | 2 | |
| s t r l k e | 2 | |
| n e w e s t | 1 | |

# Frequently Adjacent Symbol Pair Merging

- To add a new token to the vocabulary, first, we identify the **most frequent symbol pair.**
- Then we **merge that most frequent symbol pair** and add it to the vocabulary.

| Character Sequence | Frequency | Vocabulary |
|---|---|---|
| M o st | 3 | m,o,s,t,e,w,r,l,k,n,st |
| W e st | 3 | |
| st e m | 2 | |
| st r l k e | 2 | |
| n e w e st | 1 | |

- By looking at above table we can see that **most frequent symbol pair** is **"st"** which occur (3+3+1=)7 times. Thus, **"st"** is added in the vocabulary list.

# Repeat the Merging Step

- We **repeat the character merging step iteratively** until we reach the vocabulary size.

| Character Sequence | Frequency | Vocabulary |
|---|---|---|
| M o st | 3 | m,o,s,t,e,w,r,l,k,n,st,we |
| We st | 3 | |
| st e m | 2 | |
| st r l k e | 2 | |
| n e we st | 1 | |

- To proceed further we can see that **“we” and “st”** tokens are highly used 4 times so we can add **“west”** in the vocabulary list.

| Character Sequence | Frequency | Vocabulary |
|---|---|---|
| M o st | 3 | m,o,s,t,e,w,r,l,k,n,st,we,west |
| West | 3 | |
| st e m | 2 | |
| st r l k e | 2 | |
| n e west | 1 | |

- Finally, we are having the vocabulary list length is 13 so we can stop repeating this process.
- BPE vocabulary list will be [m,o,s,t,e,w,r,l,k,n,st,we,west]

# BPE Algorithm

1. Extract the words from the given dataset along with their count.
2. Define the vocabulary size.
3. Split the words into a character sequence.
4. Add all the unique characters in our character sequence to the vocabulary.
5. Select and merge the symbol pair that has a high frequency.
6. Repeat step 5 until the vocabulary size is reached.

# BPE Encoding Example

**BPE vocabulary** : [m, o, s, t, e, w, r, l, k, n, st, we, west]

- Let's assume our input text consist word **"East"**.
- We will check whether this word is present in vocabulary list or not.
  - As the word is not present in vocabulary list, we will split this word in subwords, which become **["Ea","st"]**.
  - As **"st"** is present in the vocabulary list, we will further split word in the subwords.
  - Thus, we will produce tokens **["e","a","st"]**.
- As **"a"** is not present in our vocabulary list and further splitting is not possible out final token list will be like this.
  - **["e",< UNK >,"st"]**
  - As we have taken very small corpus that's why we are getting <UNK> token. When we deal with huge corpus our vocabulary list will be having all the characters.

# BPE: Assessment & Use Cases

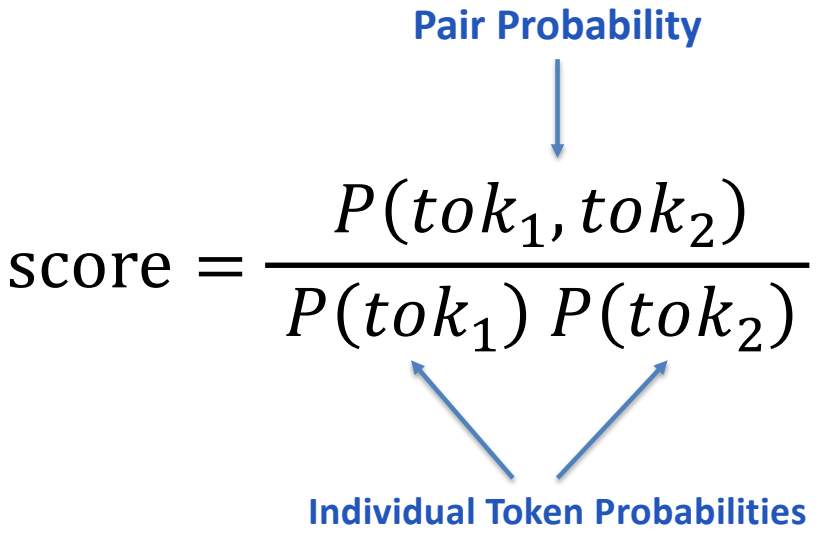
## Standard for Generative AI (GPT, Gemini, LLaMA)

| Advantages | Limitations |
| --- | --- |
| • **Adaptive Vocabulary:** Learned directly from training data.<br>• **UNK Killer:** Effectively handles OOV words by breaking them down.<br>• **Compression:** Optimizes sequence efficiency with variable length tokens. | • **Pre-tokenization Dependency:** Requires splitting by whitespace first.<br>• **A Friction with Non-Latin:** Difficult for languages like Japanese/Chinese.<br>• **Greedy Heuristic:** Not linguistically perfect, just statistically frequent. |

# WordPiece

Optimizing for likelihood, not just frequency.

- **Used By:** BERT, Google.
- **The Difference:** Instead of merging the most frequent pair, WordPiece merges the pair that maximizes the **likelihood** of the language model's training data.
- It optimizes for information gain.



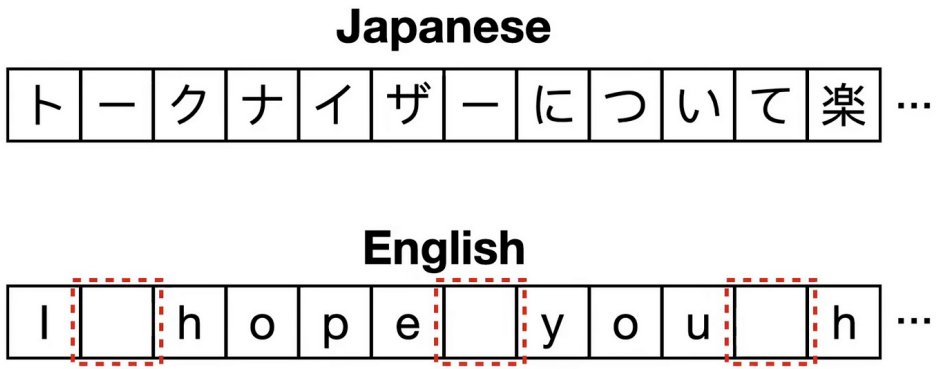
# WordPiece Mechanics


- **Prefix System:** Subword units that are not at the beginning of a word get a special prefix (e.g., ##). This tells the model these pieces attach to the previous token.
- **Pros:** Highly effective for Masked Language Modeling (BERT).
- **Cons:** Inherits pre-tokenization limitations (needs spaces).

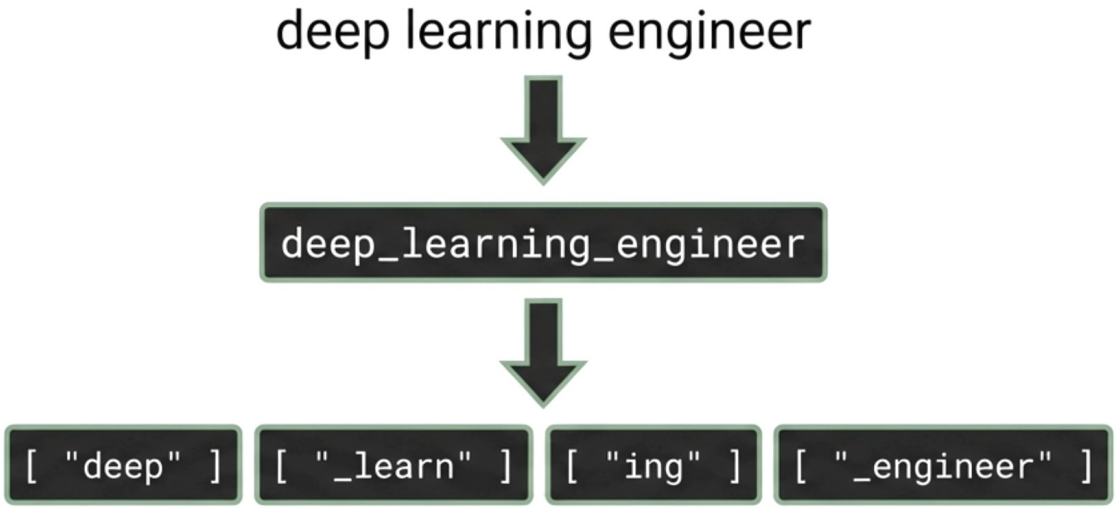
# SentencePiece

## The language-agnostic universal tokenizer.

- **Used By:** T5, ALBERT, Multilingual models.
- **The Innovation:** Treats input as a raw stream of characters.
- **No Pre-tokenization:** It does not need to split by space first, solving the problem for non-spacing languages.



# Treating space as a character




- **Lossless Reconstruction:** Because spaces are preserved as original text can be perfectly restored.

# The Showdown: Comparing the Algorithms

| Feature | BPE | WordPiece | SentencePiece |
|---|---|---|---|
| Pre-Tokenization | Required (Split by space) | Required | **Not Required** |
| Merging Strategy | Most Frequent Pairs | Likelihood Maximization | BPE or Unigram Model |
| Space Handling | Separator | Separator | **Included as Token (_)** |
| Primary Use Case | GPT, Llama | BERT | T5, Multilingual |

# BPE Code Example

- BPE is used as a preprocessing tokenization technique in GPTx models. Let's demonstrate this using the Hugging Face tokenizers library, which offers a variety of tokenizers including BPE.

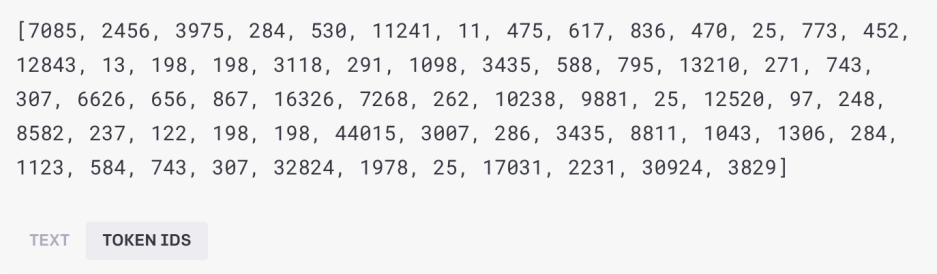
```
from tokenizers import Tokenizer
from tokenizers.models import BPE
from tokenizers.pre_tokenizers import Whitespace
from tokenizers.trainers import BpeTrainer

# Create a BPE tokenizer
tokenizer = Tokenizer(BPE())
tokenizer.pre_tokenizer = Whitespace()

# Trainer to train the tokenizer
trainer = BpeTrainer(special_tokens=["[UNK]", "[CLS]", "[SEP]", "
[PAD]", "[MASK]"])

# Example corpus to train the tokenizer
corpus = ["unbelievable", "belief", "believable", "unbelievably"]

# Train the tokenizer
tokenizer.train_from_iterator(corpus, trainer)

# Tokenize a text
output = tokenizer.encode("unbelievable")
print(output.tokens)
```

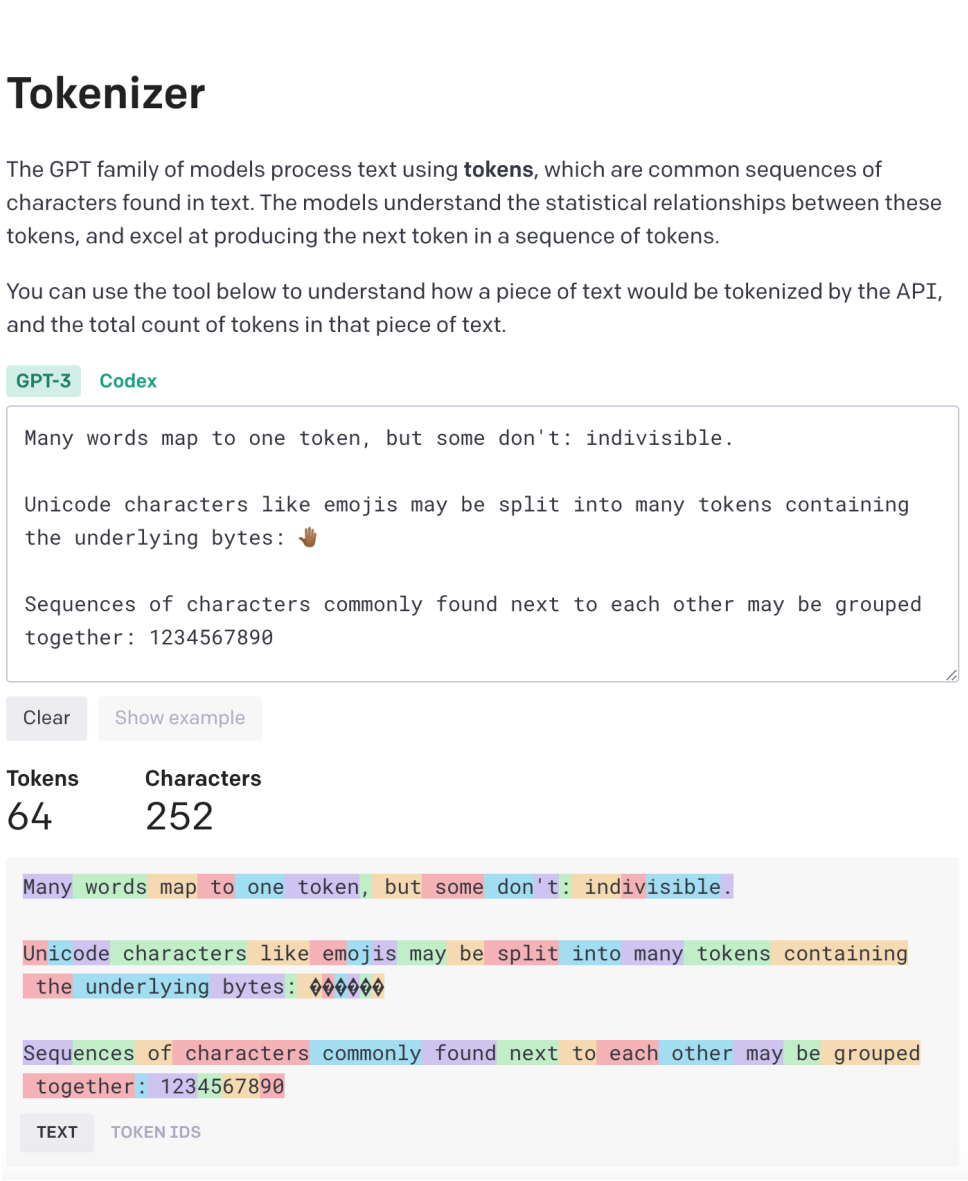
https://vitalflux.com/nlp-tokenization-in-machine-learning-python-examples/

# OpenAI Tokenizer

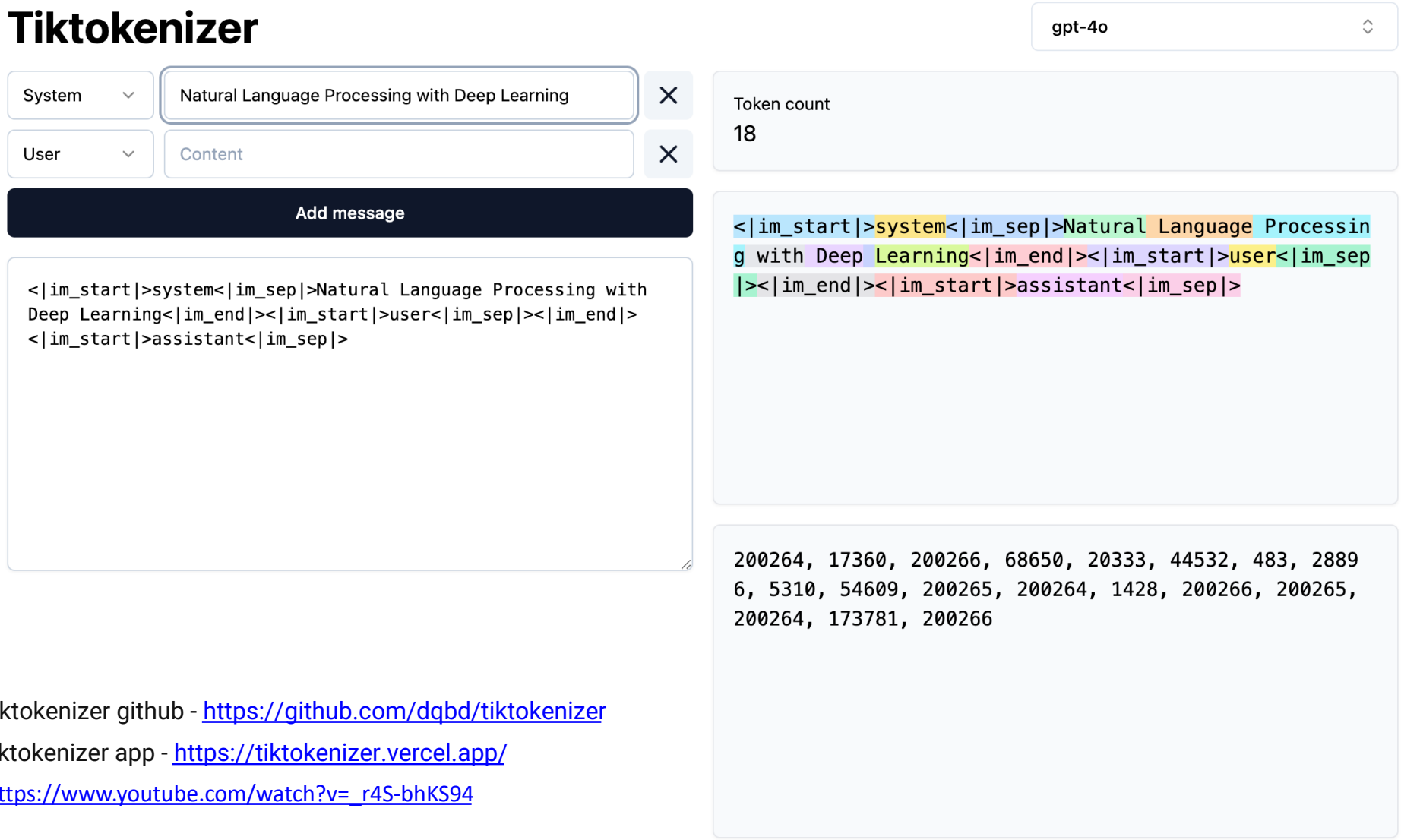
**Tiktoken** is a fast BPE tokeniser for use with OpenAI's models.

- https://github.com/openai/tiktoken
- https://platform.openai.com/tokenizer



# Tiktokenizer




tiktokenizer github - https://github.com/dqbd/tiktokenizer

tiktokenizer app - https://tiktokenizer.vercel.app/

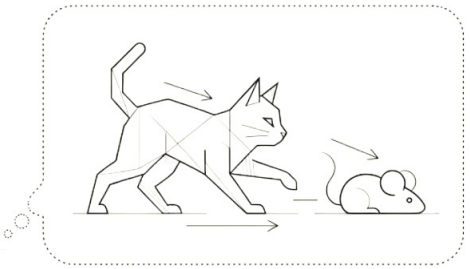
https://www.youtube.com/watch?v=_r4S-bhKS94

# Word Embeddings

# What Are Word Embeddings?

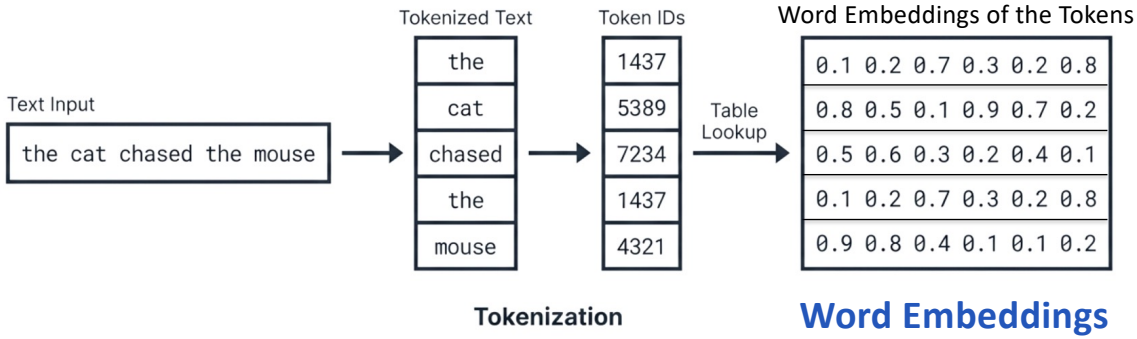
## Neural Networks Do Not Read Text. They Process Vectors.

### Human Perception

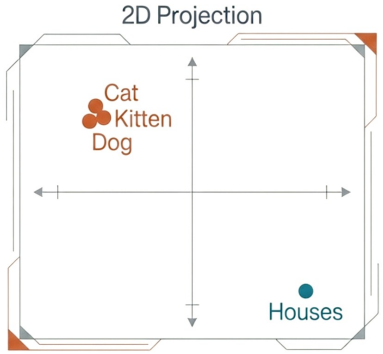


Humans naturally understand sentences like “**the cat chased the mouse**,” picturing the action and word relationships.

### Machine Representation




- Computers process **numbers.**
- **Embeddings** act as the bridge, converting text into a vector format machines can manipulate mathematically.
- The goal is a space where semantic distance mirrors mirrors human understanding.



# How Do Word Embeddings Work?

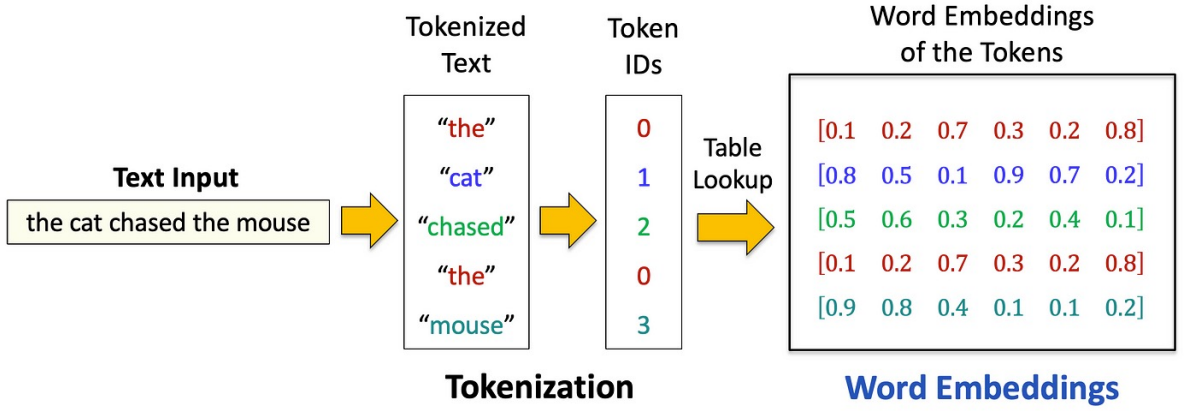
- **Core Idea**: Words are mapped to numerical vectors (**dense vectors**).
- **Vocabulary**: Predefined set of recognized words from tokenization.
- Each token gets a unique vector in a lookup table.
  - **Vocabulary**: ["the", "cat", "chased", "mouse"]
  - **Token IDs**: [0, 1, 2, 3]
  - **Lookup Table**:

    0 ("the"): [0.1, 0.2, 0.7, 0.3, 0.2, 0.8]

    1 ("cat"): [0.8, 0.5, 0.1, 0.9, 0.7, 0.2]

    2 ("chased"): [0.5, 0.6, 0.3, 0.2, 0.4, 0

    3 ("mouse"): [0.9, 0.8, 0.4, 0.1, 0.1, 0.2]

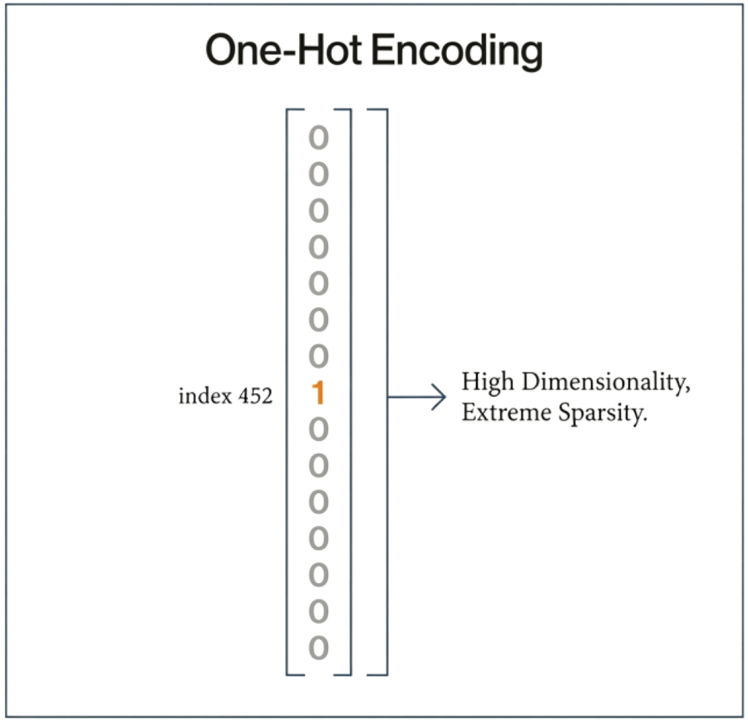


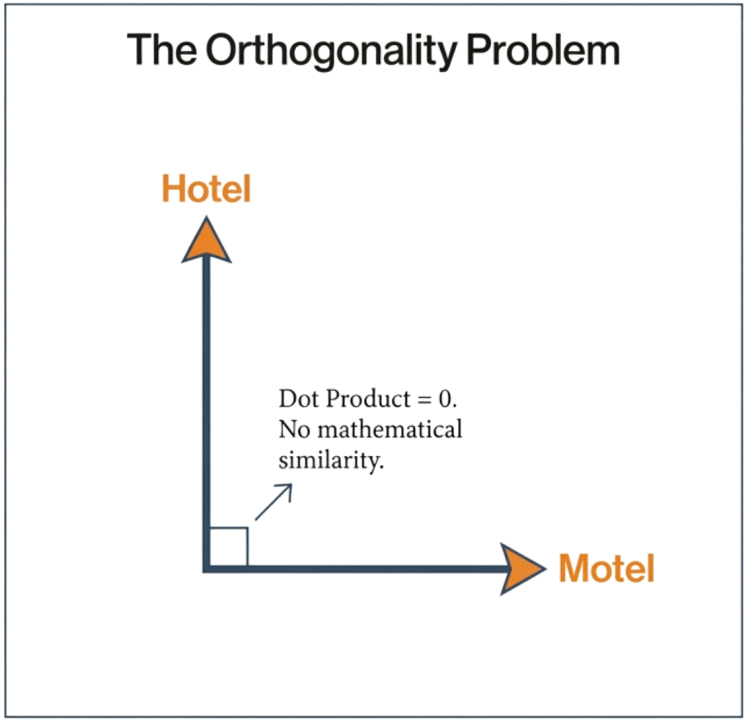
# How Are Word Vectors Created?

- At the heart of word embeddings lies a fundamental question: **How do we generate these vectors?**
- This journey has evolved dramatically over decades, from basic techniques to advanced neural methods. Below, we trace this progression:
  - **One-hot vectors :** Simple but limited in capturing meaning.
  - **Distributional Hypothesis**: N-Gram and Count-Based Methods
  - **Neural Word Embeddings**: NPLM, Word2Vec, GloVe, and FastText – Advanced models that capture linguistic context and relationships.
- Each milestone represents a leap forward in encoding language meaning and context.

# Stage 1: The Era of Isolation

Represent each word using a vector of size of the vocabulary.



# Shortcomings of One-Hot Encoding

- **High dimensionality**: One-hot vectors are sparse and high-dimensional, with the vector size equal to vocabulary size.
- **No semantics**: One-hot encoding treats each word independently, encoding no semantic information about relationships between words. All vectors are orthogonal.
  - the, cat
  - cat, chased
  - cat, mouse

Vectors of any pair of words will have a dot product of zero
(i.e. show zero similarity)

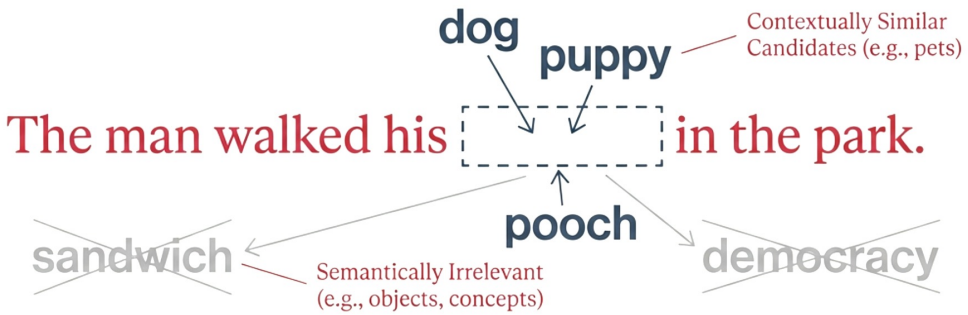
$$\text{the} \cdot \text{cat} = [1,0,0,0] \cdot [0,1,0,0]^T = 0$$

$$\text{cat} \cdot \text{mouse} = [1,0,0,0] \cdot [0,0,0,1]^T = 0$$

# The Theoretical Shift: Distributional Hypothesis

**“You shall know a word by the company it keeps.”**

- J.R. Firth (1957)




- **Zellig Harris‘ Hypothesis**: Words used in close proximity likely have similar meanings.

  近朱者赤，近墨者黑

- **The Insight**: We don't need to manually define a word; we can define it by its neighbors. This simple idea is the foundation of modern NLP.

# Context is King: The Lychee Example.

A bowl of **lychees** is on the **dining table**.

Everyone loves the **sweet taste** of **lychees**.

**Lychees** are often served chilled during **summer**.

---

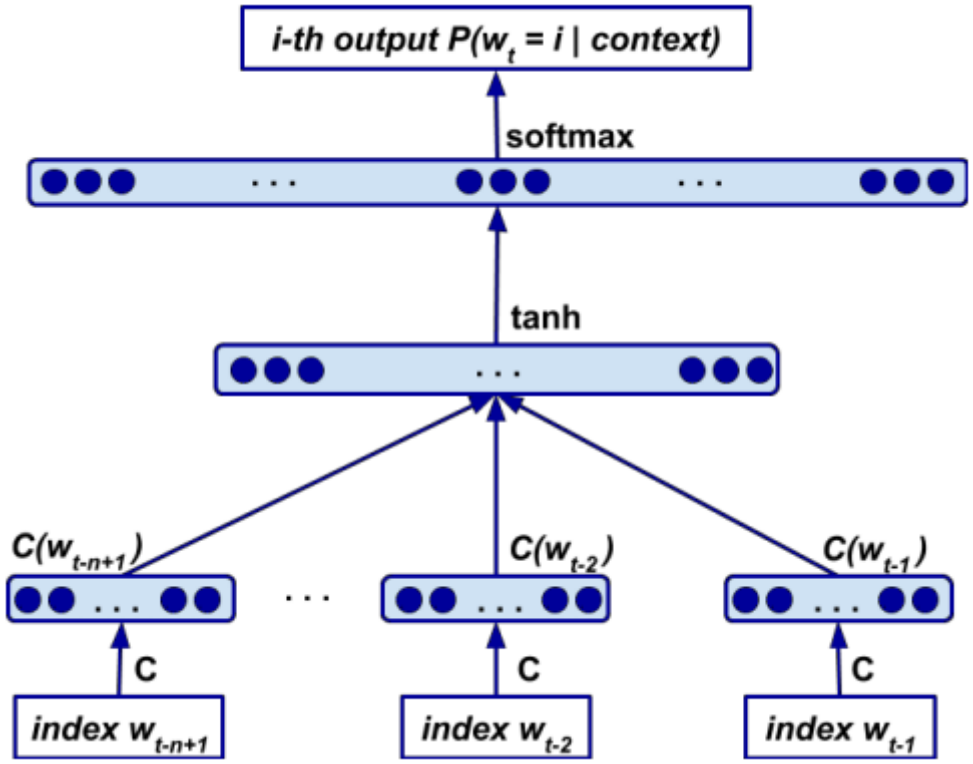
**Inference:** Even without a definition, the context reveals that a ‘lychee’ (荔枝) is a sweet, refreshing fruit eaten in summer.

**Words used in close proximity likely have similar meanings.** 近朱者赤，近墨者黑

- If a machine sees 'Lychee' appearing in the same context as 'Apple' or 'Grape,' it pushes their vectors closer together.

# The Neural Revolution: NPLM (2003)

## Reduced the Dimensions.

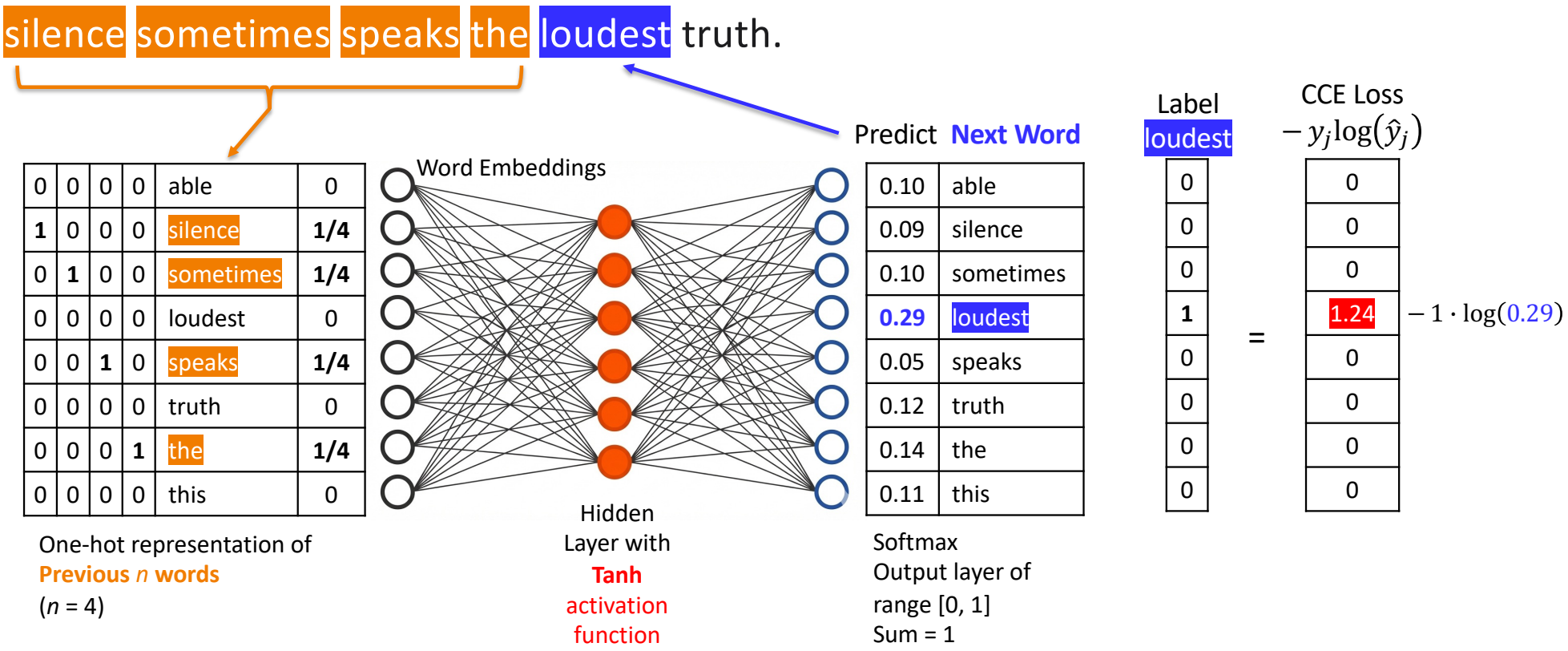



**The Breakthrough:** Yoshua Bengio et al. introduced the **Neural Probabilistic Language Model (NPLM)**.

**End-to-End Learning:** For the first time, a neural network learned language modeling and word embeddings simultaneously.

**Significance:** This moved us from sparse, isolated vectors to dense, trained representations.

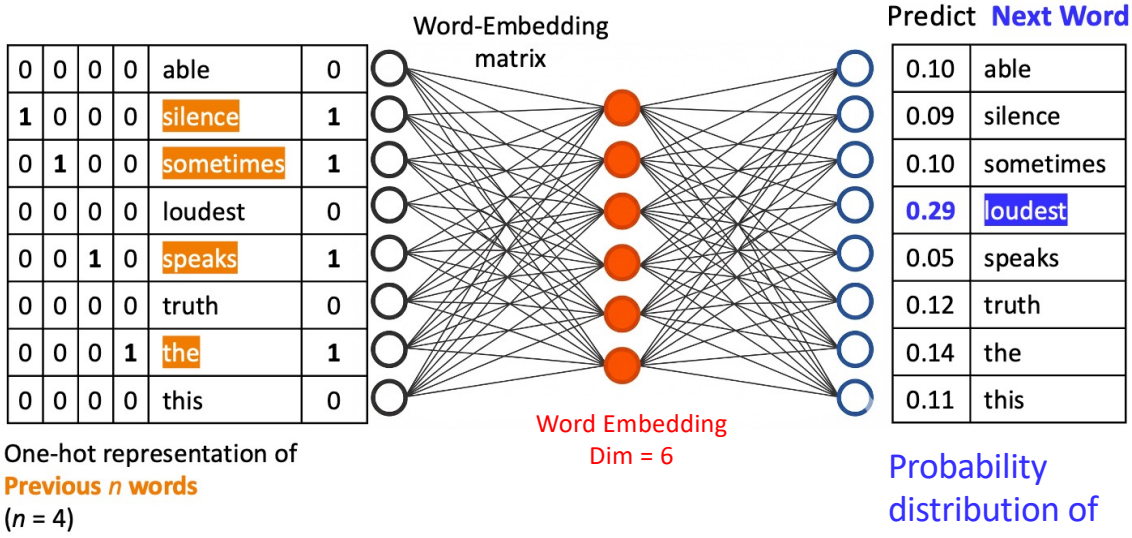
# NPLM Architecture: Autoregressive Language Model




The NPLM takes the average of the one-hot encodings of the **previous *n* words** as input and predicts the one-hot vector of the **next word**, trained with categorical cross-entropy (CCE) loss.

# NPLM Impact

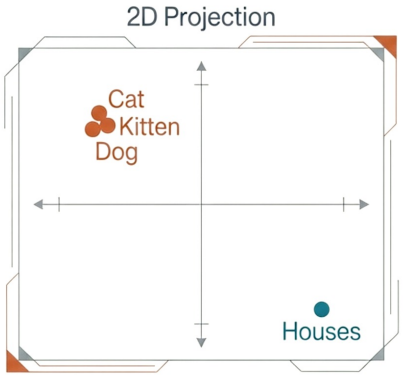
silence sometimes speaks the loudest truth.




**Architecture:** Input words are converted to dense vectors. The hidden layer captures context (e.g., linking " silence sometimes speaks the to loudest“).

**Solving the Curse of Dimensionality:** Instead of vectors size 100,000 (One-Hot), NPLM uses manageable sizes (e.g., 64, 128, 256).

**Result:** Words with similar meanings automatically cluster together in vector space.



# Word2Vec (2013)

**Efficient Estimation of Word Representations in Vector Space**

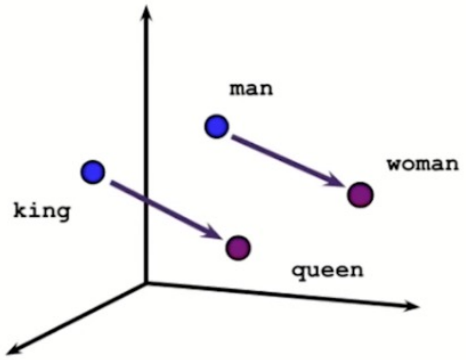
https://arxiv.org/pdf/1301.3781.pdf

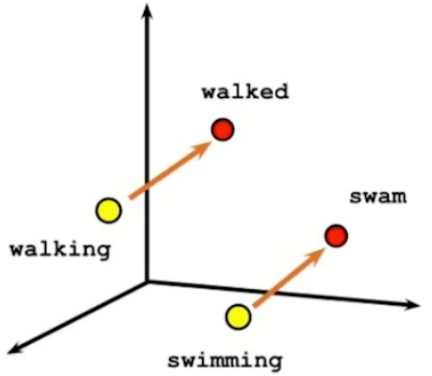

**Tomas Mikolov**
Google Inc., Mountain View, CA
tmikolov@google.com

**Kai Chen**
Google Inc., Mountain View, CA
kaichen@google.com

**Greg Corrado**
Google Inc., Mountain View, CA
gcorrado@google.com

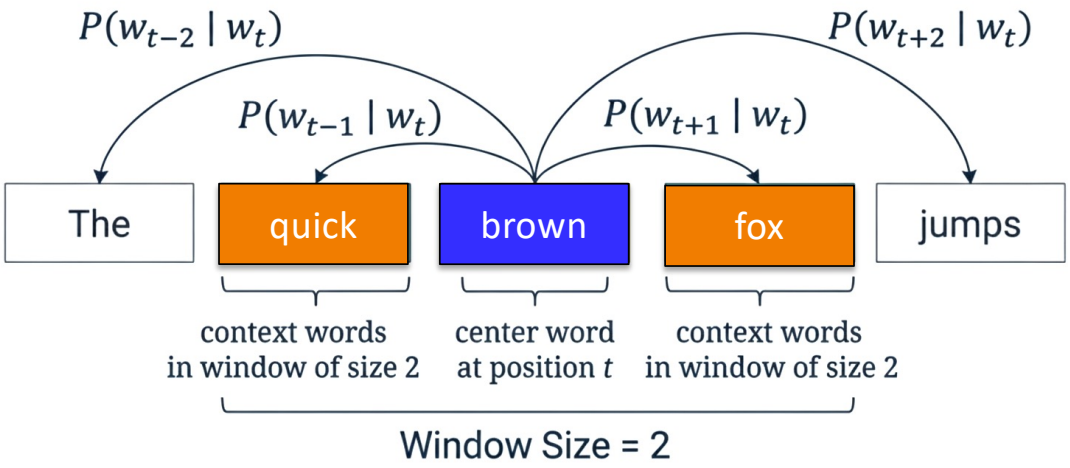
**Jeffrey Dean**
Google Inc., Mountain View, CA
jeff@google.com


king – queen ~= man – woman

# Generating Training Data: The Sliding Window




Target word: brown
Context words: quick and fox

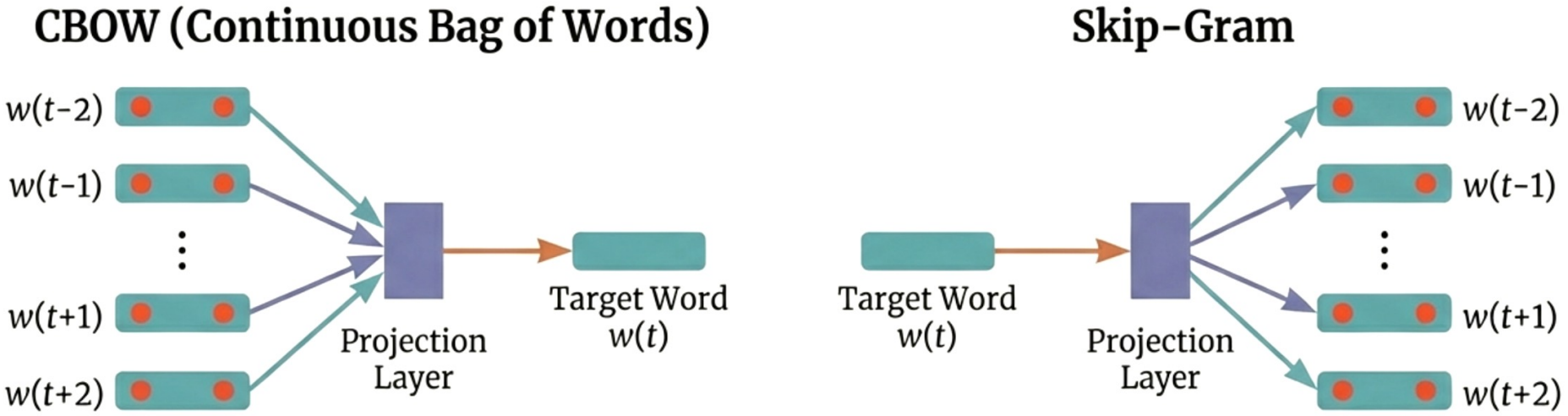
Window size = 2

- **The Setup:** We define a "Window Size" (e.g., 2 words left/right).
- **The Data Pairs:** The network trains on pairs like (brown, [quick, fox]).
- This simple mechanical process turns raw text into a supervised learning problem.

# Word2Vec (2013): A Paradigm Shift

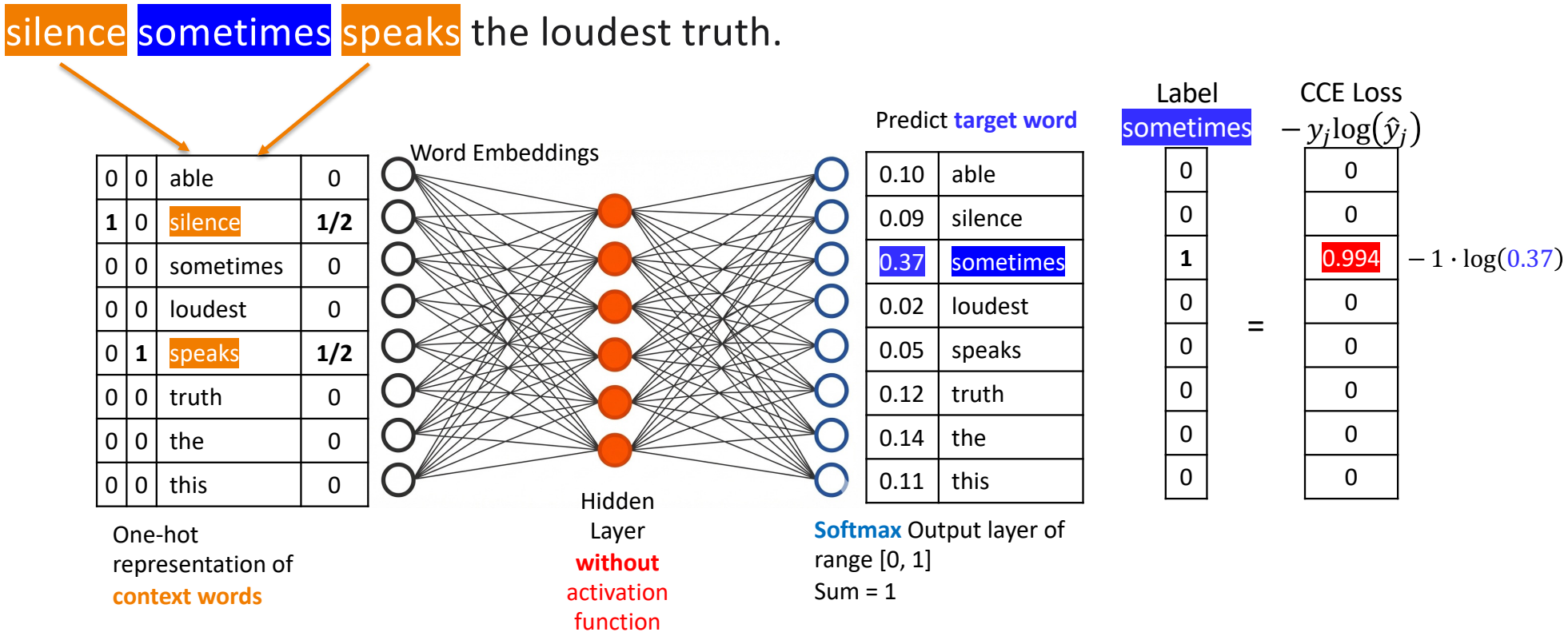
## Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean (Google Inc.)




In 2013, Google introduced Word2Vec. It moved beyond simple language modeling to a highly efficient framework for learning dense vector representations from massive datasets.

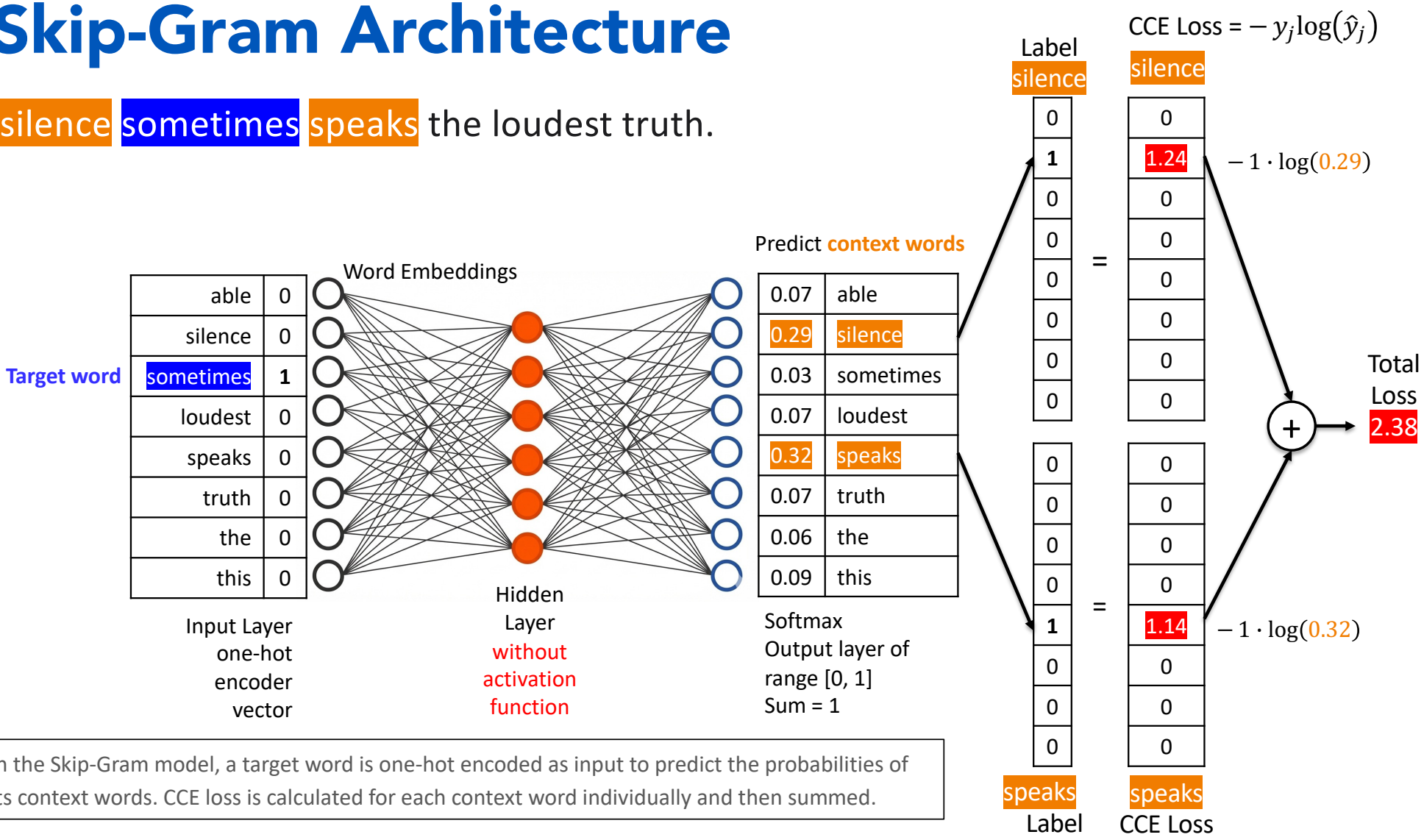
# Continuous Bag of Words (BOW)




In the CBOW model, the average of the one-hot encodings of the context words is used to predict the one-hot encoding of the target word.

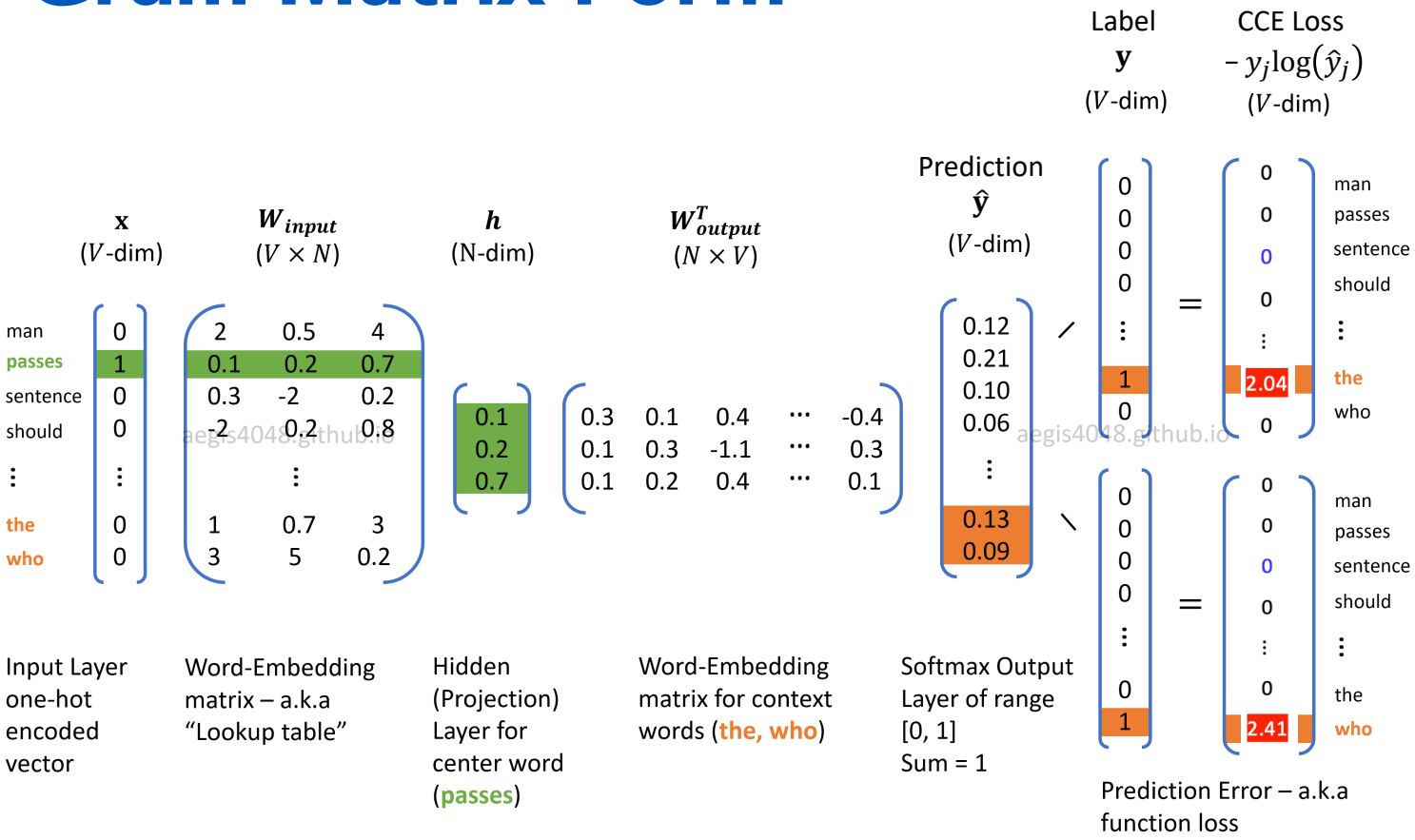
# Skip-Gram Architecture

silence sometimes speaks the loudest truth.




In the Skip-Gram model, a target word is one-hot encoded as input to predict the probabilities of its context words. CCE loss is calculated for each context word individually and then summed.

# Skip-Gram Matrix Form




https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

# The Computational Bottleneck

$$\text{softmax}(z_i) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

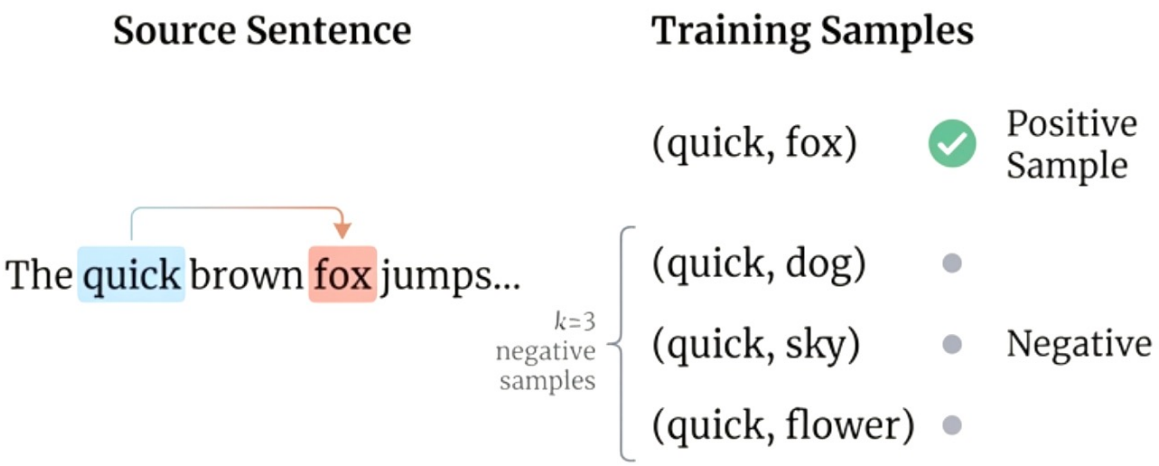
Heavy Computation (Summing over 100,000+ words per step).

**The Problem**: To predict a word, standard Softmax must calculate the probability against every other word in the vocabulary to ensure they sum to ensure they sum to 1.

**The Scale:** If Vocabulary = 100,000, the model performs 100,000 calculations for every single training step.

**The Verdict:** Computationally impractical for large-scale training

# Optimization: Negative Sampling



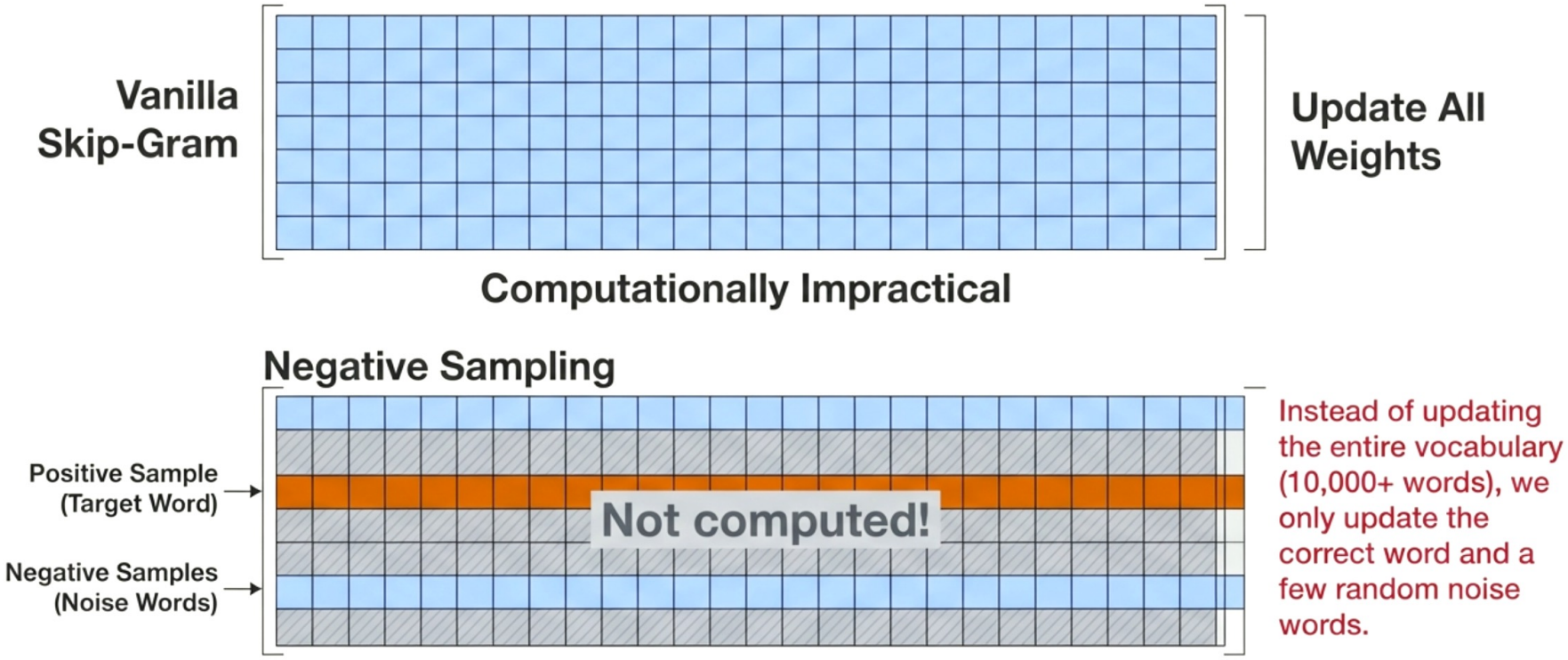

**The Solution:** Instead o f updating weights for the entire vocabulary, we only update a tiny fraction.

**How it Works:** We update the correct answer (Positive plus a small number (k=5 to 20) of random, unrelated words (Negative).

**The Result:** The model learns to distinguish the 'real' context from random noise, drastically reducing computational load.

# Optimizing for Speed: Negative Sampling

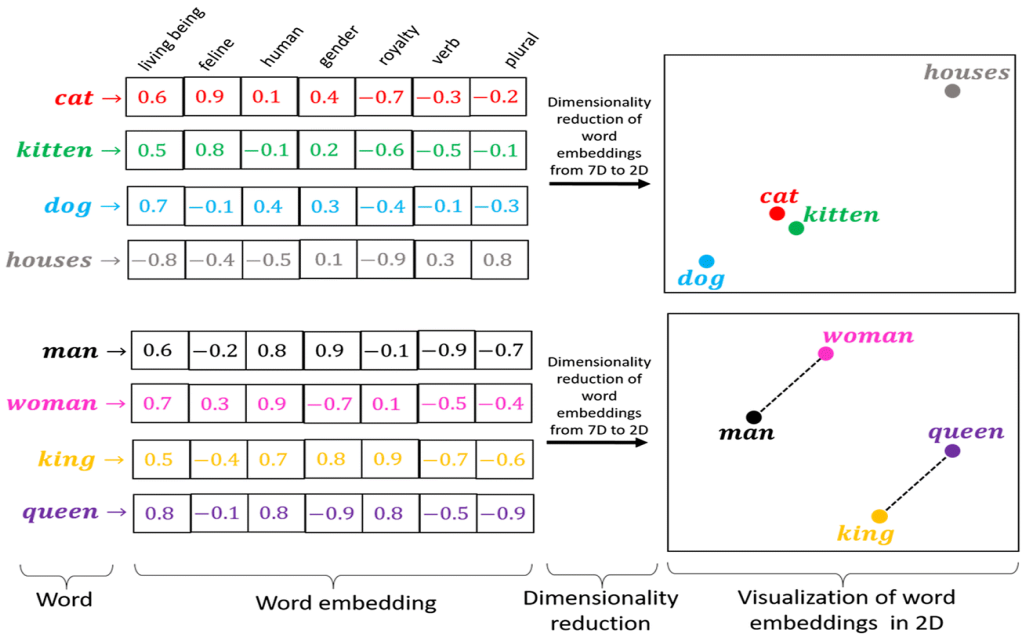
## Solving the computational bottleneck.



# Advantages of Word2Vec

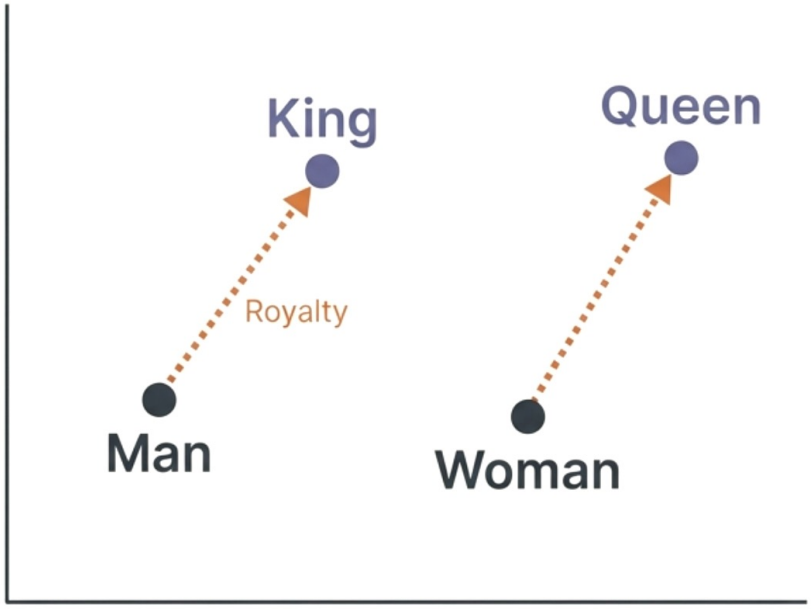
- Overcomes curse of **dimensionality (256 – 768) with simplicity**
- Solve the issue of **Synonymy** (同义字)
  - Different words with similar meaning closer in embedding space.

- Training data can be easily obtained from raw text and models can be trained by self-supervised learning.
- Pretrained embeddings used in a host of applications



# Semantic Arithmetic: The "King - Man" Equation

Encoding concepts as directions.




**King – Man + Woman ≈ Queen**

The most famous result of Word2Vec is its ability to perform algebraic operations on concepts.

- **The Meaning:** The vector path from "Man" to "King" represents the concept of "Royalty." Applying that same path to "Woman" lands you on "Queen."
- This proves the model has captured semantic relationships, not just statistics.

# Colab: Word2Vec Demo

```
[56] import gensim.downloader

     # model= gensim.downloader.load('word2vec-google-news-300')
     word2vec_model = gensim.downloader.load('glove-wiki-gigaword-50')
```

```
[59] # Get the embedding vector for a word
     cat_embedding = word2vec_model["cat"]
     kitten_embedding =word2vec_model["kitten"]
     hourse_embedding = word2vec_model["houses"]
```

```
[60] cat_embedding.shape

     (50,)
```

```
     cat_embedding

     array([ 0.45281 , -0.50108 , -0.53714 , -0.015697,  0.22191 ,  0.54602 ,
            -0.67301 , -0.6891  ,  0.63493 , -0.19726 ,  0.33685 ,  0.7735  ,
             0.90094 ,  0.38488 ,  0.38367 ,  0.2657  , -0.08057 ,  0.61089 ,
            -1.2894  , -0.22313 , -0.61578 ,  0.21697 ,  0.35614 ,  0.44499 ,
             0.60885 , -1.1633  , -1.1579  ,  0.36118 ,  0.10466 , -0.78325 ,
             1.4352  ,  0.18629 , -0.26112 ,  0.83275 , -0.23123 ,  0.32481 ,
             0.14485 , -0.44552 ,  0.33497 , -0.95946 , -0.097479,  0.48138 ,
            -0.43352 ,  0.69455 ,  0.91043 , -0.28173 ,  0.41637 , -1.2609  ,
             0.71278 ,  0.23782 ], dtype=float32)
```

```
[66] # Compute the Distance between two word embeddings
     word2vec_model.distance('cat','kitten')

     0.36136943101882935
```

```
[67] # Show the distance between cat and houses is larger than cat and kitten
     word2vec_model.distance('cat','houses')

     0.7752506732940674
```

```
# Find the most similar word of cat
word2vec_model.most_similar('cat')
```

```
[('dog', 0.9218006134033203),
 ('rabbit', 0.8487821221351624),
 ('monkey', 0.8041081428527832),
 ('rat', 0.7891963124275208),
 ('cats', 0.7865270972251892),
 ('snake', 0.7798910737037659),
 ('dogs', 0.7795814871788025),
 ('pet', 0.7792249917984009),
 ('mouse', 0.773166835308075),
 ('bite', 0.7728800177574158)]
```

```
# Find the most similar word of king
word2vec_model.most_similar('king')[0]
```

```
('prince', 0.8236179351806641)
```

```
# Word Embedding Arithmetic "king" + "woman" - "man" = "queen"

sims = word2vec_model.most_similar(positive=['king', 'woman'], negative=['man'])

sims
```

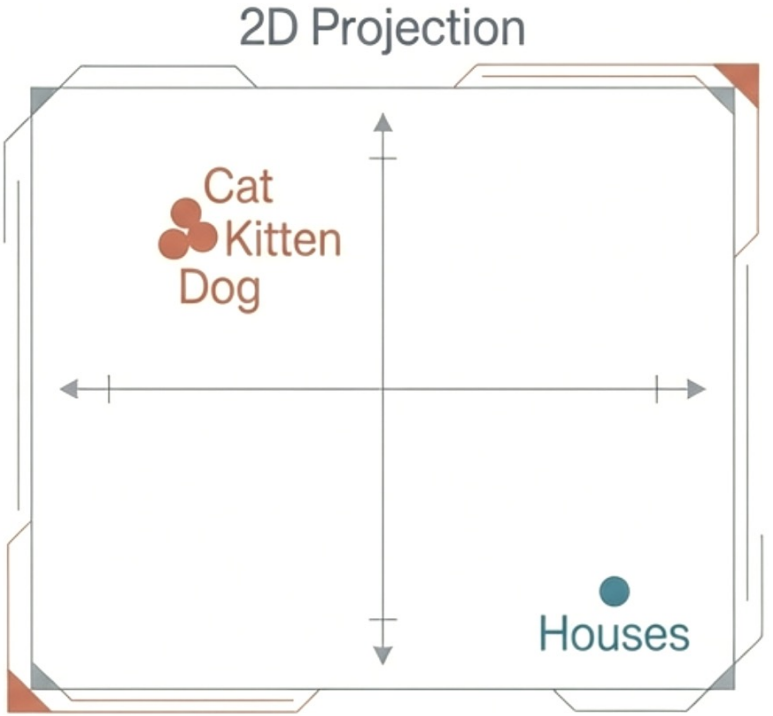
```
[('queen', 0.8523604273796082),
 ('throne', 0.7664334177970886),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883628845215),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337412238121033),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184861898422241),
 ('widow', 0.7099431157112122)]
```

queen = king + woman - man

https://colab.research.google.com/drive/1wT-OGqZy_6o3Q_FKXWIHqG87A1åiLSNGr?usp=sharing

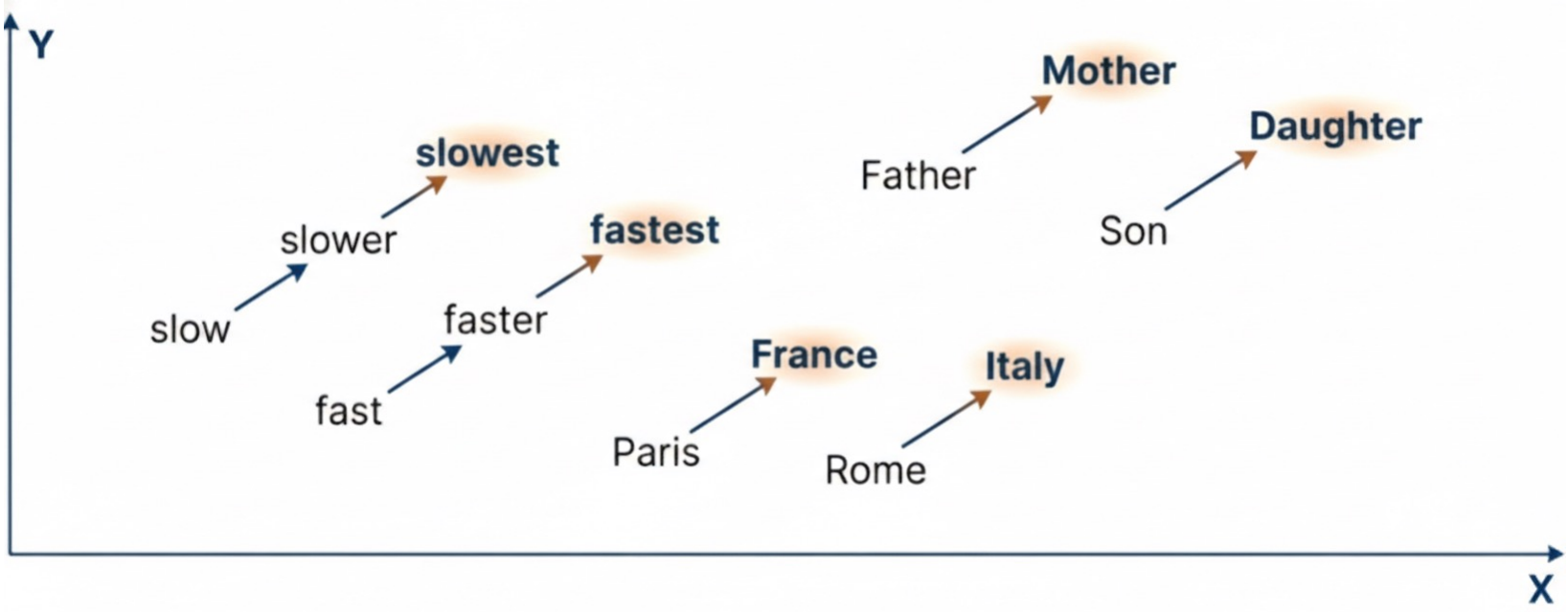
# Visualizing the Semantic Space (1)

| Word | Feline | Human | Object |
|---|---|---|---|
| Cat | **0.9** | 0.1 | -0.4 |
| Kitten | **0.8** | -0.1 | -0.5 |
| Houses | -0.4 | -0.5 | **0.8** |




Dimensionality Reduction compresses 7D vectors to 2D for visibility, preserving clusters.

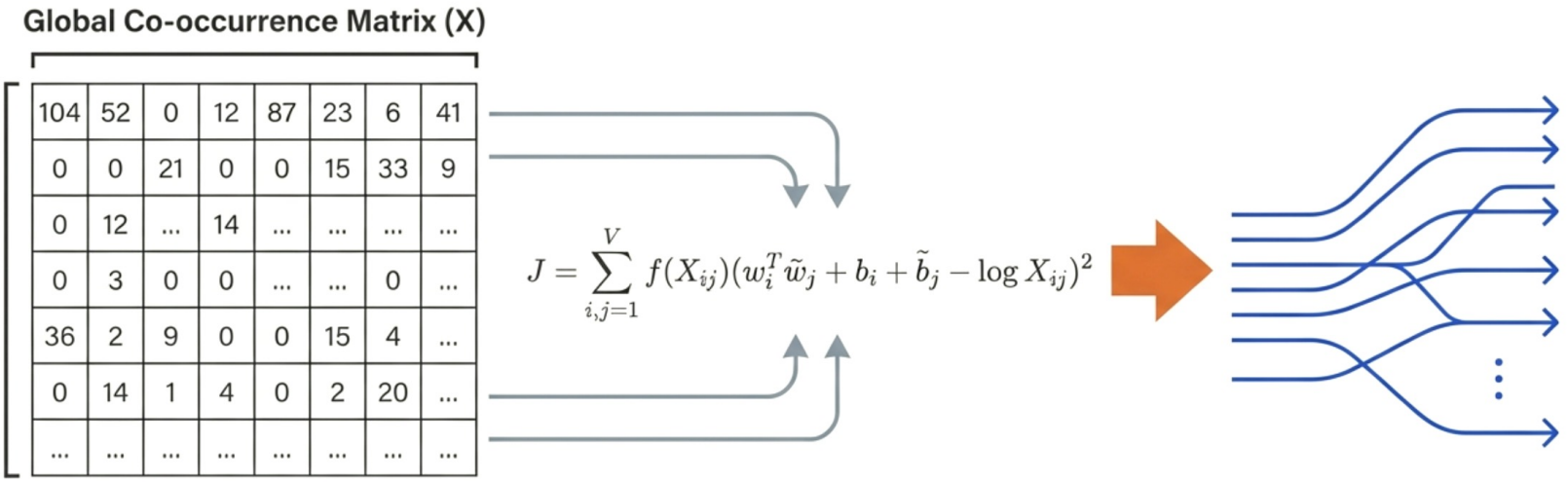
# Visualizing the Semantic Space (2)




Word2Vec organizes words so that linguistic patterns are preserved. Superlatives, capitals, and family relationships all form consistent geometric structures.

# Expanding the View: GloVe (2014)

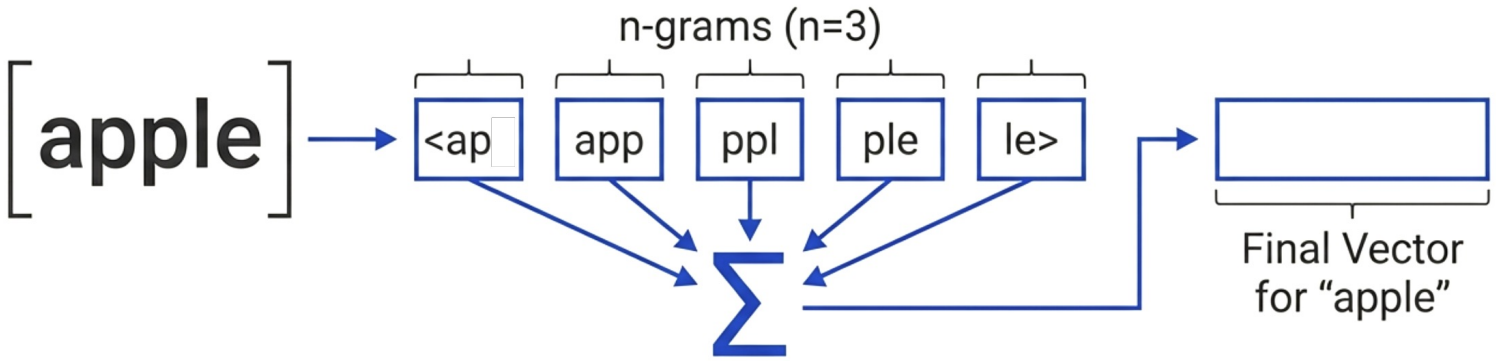
## Global Vectors for Word Representation.




- Unlike Word2Vec which learns window-by-window, GloVe calculates statistics across the entire corpus at once using a global matrix.

# Handling the Unknown: FastText (2015)

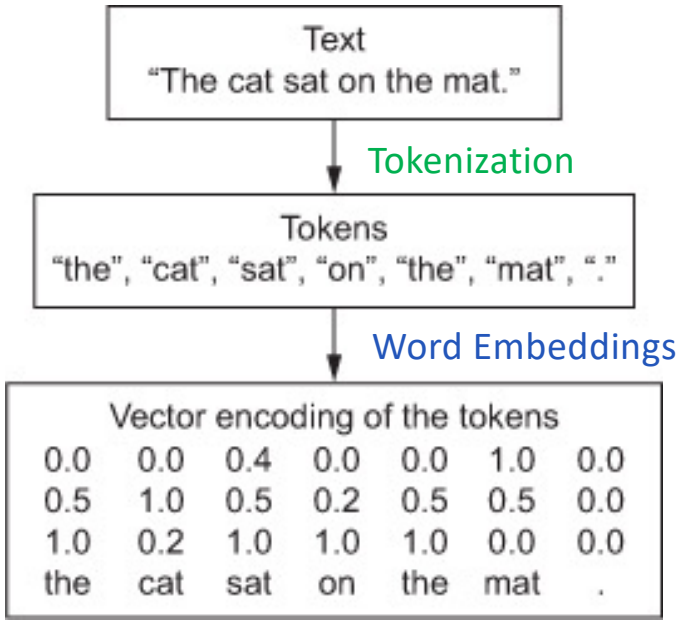
## Sub-word information and n-grams.




- By breaking words into chunks (n-grams), FastText can construct meanings for unknown words or typos (e.g., "appple") by summing the vectors of known parts.
- **This boosts robustness for out-of-vocabulary words** and excels in morphologically rich languages like Finnish, enhancing Skip-Gram’s generalization for real-world use.

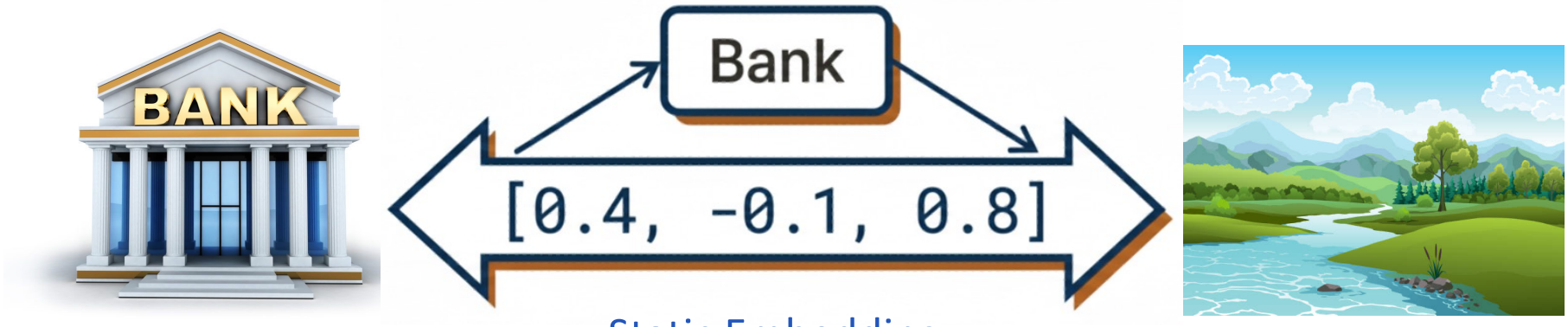
# Tokenization and Word Embedding

- **Tokenization splits text into tokens** (sub-words)
- **Word Embeddings convert tokens into low-dimensional numerical vector representations** with semantic meaning
  - Word embeddings allow NLP models to utilize semantic similarities, effectively addressing the problem of **Synonymy** (同义词).
  - Widely used techniques like **Word2vec (CBOW and Skip-Gram)**, GolVe and FastText



# The Unsolved Problem: Polysemy (一词多义)

One vector cannot hold two meanings.



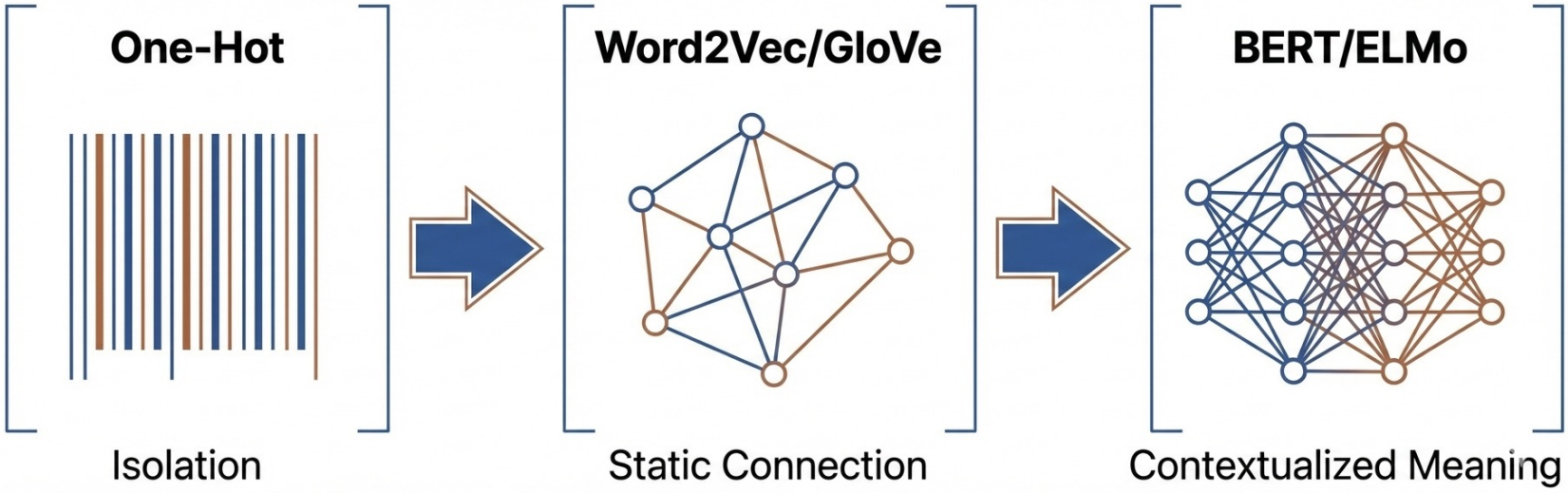

**Financial Bank**

Static Embedding
Averages both meanings

**River Bank**

Traditional embeddings like Word2Vec, assign **one static vector per word**. They cannot distinguish "River Bank" from "Financial Bank" based on context.

# The Next Leap: Contextualized Embeddings




To solve **Polysemy**, we moved to RNN (ELMo) and Transformer (BERT). In these models, the vector for ”bank" changes dynamically depending on the sentence. The bridge between human perception and machine calculation is finally complete.