Introduction to Large Language Models

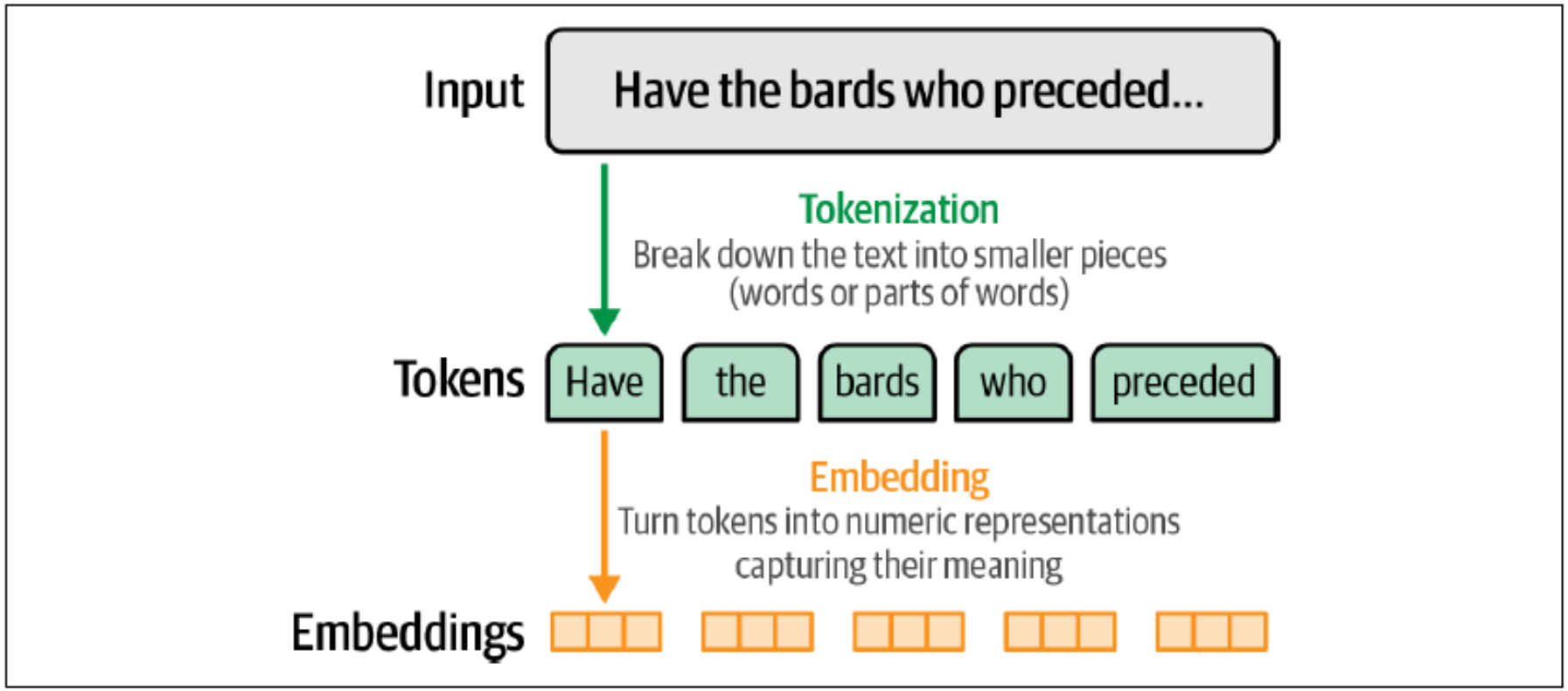
Spring 2026

Contextual Embeddings

(Some slides adapted from Ralph Grishman at NYU,
Yejin Choi at UWashington, N. Tomura at UDepaul, Jurafsky and Martin, CS224N, CS224d at Stanford and other resourses on the web)

Tokens and Embeddings

- Tokens and embeddings are two of the central concepts of using large language models (LLMs).



Source: Hands-On Large Language Models by Jay Alammar and Maarten Grootendorst

Statistical, Count-based Methods Summary

Method	Based On	Vector Type	Strengths	Weaknesses
BoW	Word counts	Sparse	Simple, fast	No semantics
TF	Normalized counts	Sparse	Document-length normalization	Still lexical only
TF-IDF	Counts + rarity	Sparse	Highlights important words	Context ignored
Co-occurrence Matrix	Word-word counts	Very sparse	Foundation for PMI/GloVe	Huge matrices
PMI	Co-occurrence probabilities	Dense or sparse	Captures association	No context handling
SVD (LSA)	Matrix factorization	Dense	Early semantic structure	Loses nuance

These methods formed the foundation for later **neural** and **contextual** embeddings like Word2Vec → GloVe → BERT → GPT.

Static Distributed / Prediction-Based Word Representations

- These are **dense vector representations** of words learned by training a neural network to **predict context**.
- They are called **static** because **each word has one fixed vector**, regardless of context.
- They are also called **distributed embeddings** because each linguistic feature is **distributed across many dimensions** of the vector.
- They are also known as **prediction-based embeddings**, in contrast to count-based embeddings like TF-IDF or co-occurrence matrices.

Main Methods:

- **Word2Vec** (CBOW, Skip-gram) https://code.google.com/archive/p/word2vec/
- **GloVe** (Global Vectors) http://nlp.stanford.edu/projects/glove/
- **FastText** (subword-enhanced) http://www.fasttext.cc/

Characteristics:

- Capture semantic similarity
- Dense vectors (50–300 dims)

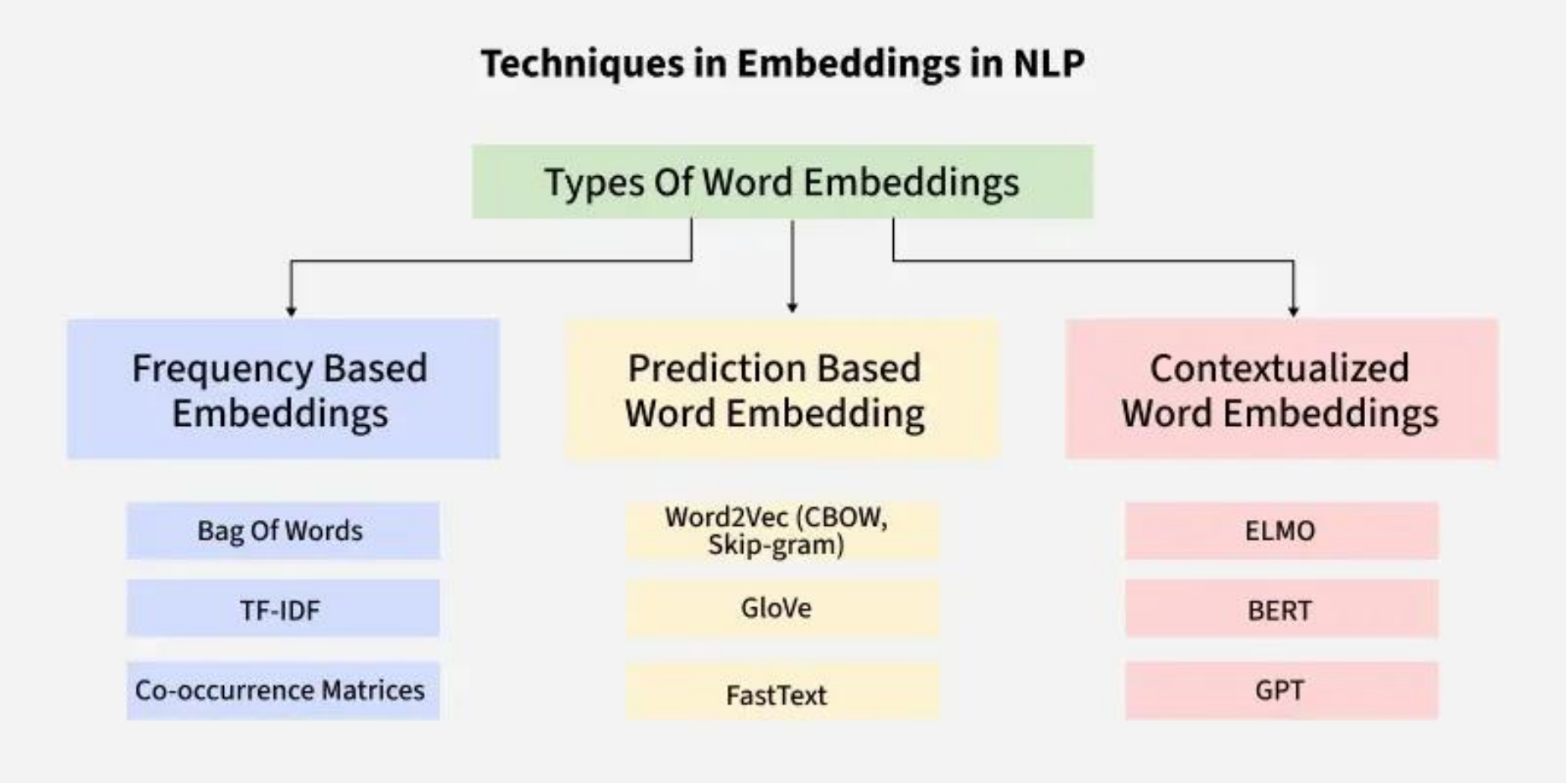
Static Distributed / Prediction-Based Word Representations

- Word2Vec, GloVe, and FastText—the three major static, distributed, prediction-based word embedding models.
- **Word2Vec vs. GloVe vs. FastText Summary**

Feature	Word2Vec	GloVe	FastText
Type	Prediction-based	Count + prediction hybrid	Prediction-based + subword
Training Objective	Predict context from word (SG) or word from context (CBOW)	Factorize global co-occurrence matrix	Predict like Word2Vec but with character n-grams
Uses Subword Info	❌ No	❌ No	✅ Yes
OOV Word Handling	Very poor	Very poor	Excellent (builds from n-grams)
Rare Word Quality	Weak	Better than W2V	Best
Global Co-occurrence	❌ Local only	✅ Global	❌ Local only
Speed	Fast	Fast	Slightly slower
Vector Quality	High	Very high	High–very high

Contextual/Contextualized Word Embeddings

- Static word embeddings were widely used **before contextual models** like: ELMo, BERT, GPT



Contextual Word Embeddings

- **Uses Deep Learning, Transformer-Based Methods.** These methods generate a **different vector for the same word depending on context**.

Example: “bank” (river) ≠ “bank” (finance)

Major Models:

- **ELMo** (bi-LSTM based)
- **BERT family** (BERT, RoBERTa, DistilBERT, etc.)
- **GPT family** (GPT-2, GPT-3, GPT-4, GPT-5 based models)
- **XLNet**
- **T5**

Characteristics:

- Dynamic, context-sensitive
- High performance in modern NLP
- Based on **Transformers**
- Provides the deepest level of language understanding, capturing sarcasm, homonyms, and syntactic nuances.

Representation learning

- **Representation learning** is a subfield of machine learning focused on automatically discovering the most useful way to represent data so that a model can perform tasks such as prediction, classification, or generation more effectively.
- Instead of relying on manually engineered features, representation learning trains algorithms to transform raw inputs (images, text, audio, tabular data) into **feature vectors** that capture the essential structure or meaning of the data.

Domain	Representation learned	Example use
Images	edges, textures, objects	object detection, segmentation
Text	semantics, syntax, meaning	chatbots, translation
Audio	phonemes, timbre, rhythm	speech recognition, music classification
Recommenders	user/item vectors	collaborative filtering, personalization

Representation learning = teaching models to discover the right features, automatically. It's foundational to nearly all modern AI and is what allows systems to understand text, images, audio, and more.

Representation Learning in Text

- Representation learning in text refers to methods that automatically transform textual input—words, sentences, documents—into dense numerical vectors that **capture their semantic and syntactic meaning**.
- These learned representations are **the backbone of modern NLP systems**: chatbots, translation models, search engines, sentiment analysis, and more.

- What do text representations capture? Well-learned text embeddings encode:
  - Semantic similarity
    "cat", "dog", "pet" → cluster together.
  - Syntactic roles
    Subject vs. object differences.
  - Context
    Meaning shifts with surrounding text.
  - Topical information
    Documents on similar topics cluster.
  - Sentiment & style (in higher-level representations)
    Positive vs. negative tone.

Contextual Embeddings

- A contextual embedding is a vector that captures how a word, phrase, or sentence means what it means in a specific context—semantically, syntactically, and pragmatically.
- The same word has different embeddings depending on surrounding words.
- Example:

  “He sat on the **bank** of the river.”

  “She went to the **bank** to get cash.”
- A contextual model (BERT, GPT) produces *two different vectors* for "bank" because the meanings differ.
- A contextual embedding for a word or sentence encodes:

Category	What the embedding represents
Meaning	Word sense matched to context
Syntax	Role in sentence structure
Semantics	Conceptual similarity & relations
Dependencies	Long-range connections, coreference
Pragmatics	Tone, sentiment, style
Prior knowledge	Patterns learned from large corpora

Positional embeddings

- **Positional embeddings** are the mechanism Transformers use to represent the **order** of tokens in a sequence.
- Because Transformers rely entirely on **self-attention**, they have **no built-in sense of sequence order** (unlike RNNs/CNNs, which process data sequentially).
  Positional embeddings *inject information about token positions* so the model can understand:
  - which word comes first,
  - which words are next to each other,
  - long-range order relationships.
- **Why do we need positional embeddings?**

  Without positional information:

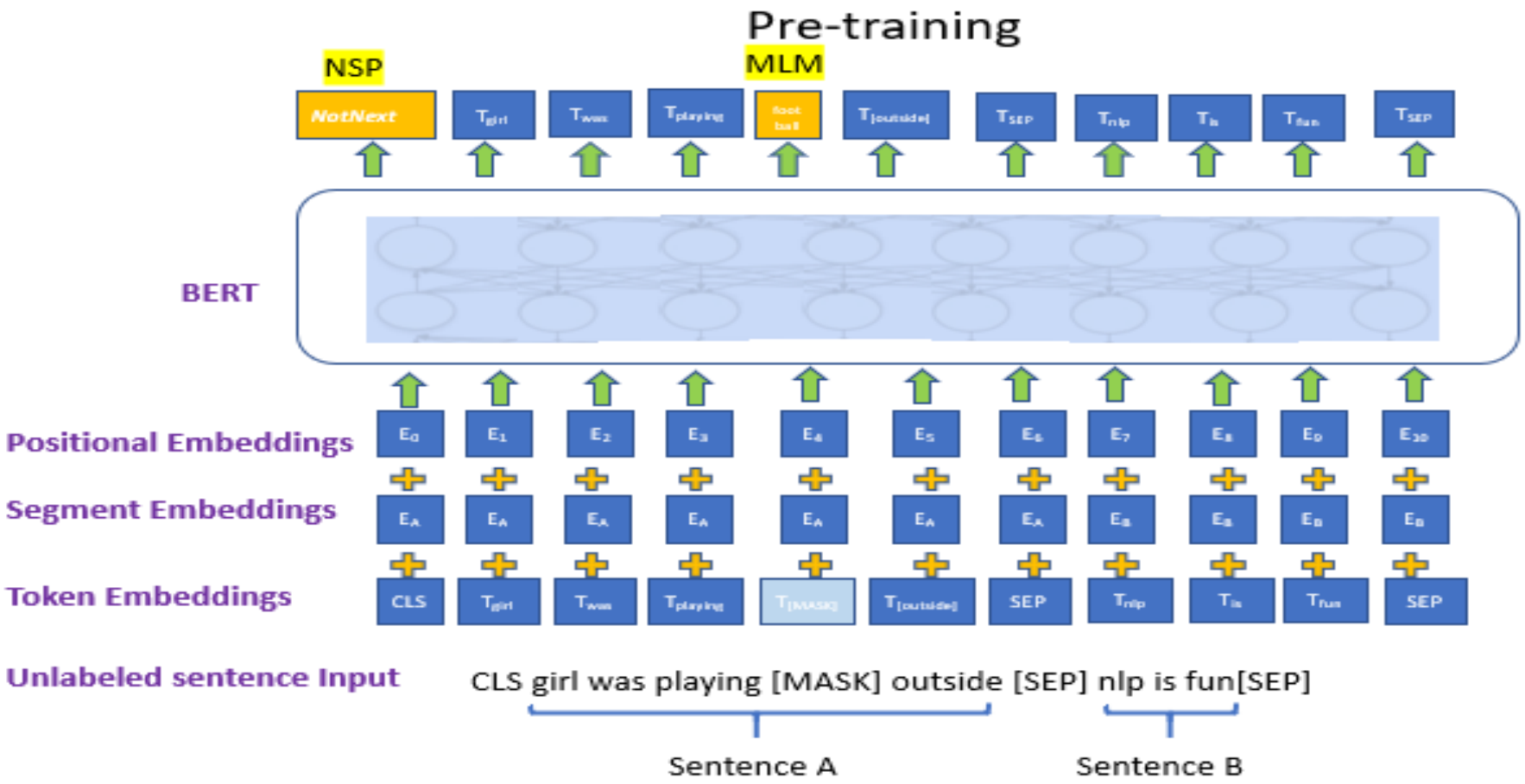
  "The cat chased the dog."

  "The dog chased the cat."

  would look identical to a Transformer because the set of tokens is the same.

- Positional embeddings prevent this by encoding **sequence structure**.

Positional embeddings in BERT



https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/

BERT Contextual Embedding Example

```
sentence = "He sat on the bank of the river. "

sentence2 = "She deposited money in the bank."
```

- First 10 elements of embedding for “bank”.
- Representations are **different**.

```
print("financial_bank elements | financial_bank elements )
print("-----------------------------------------")
for i in range(10):
    r_elem = financial_bank[i]
    f_elem = financial_bankc[i]
    print(f"{r_elem.item():<20.4f} | {f_elem.item():<20.4f}")
```

```
financial_bank element | financial_bank element copy
-----------------------------------------
0.3683                 | 0.3683
-0.6215                | -0.6215
-0.3968                | -0.3968
0.4348                 | 0.4348
0.2926                 | 0.2926
-0.0081                | -0.0081
0.2073                 | 0.2073
0.8454                 | 0.8454
0.4358                 | 0.4358
-0.8103                | -0.8103
```