Embedding Models Explained: The Reason AI Can ‘Read’ and ‘Listen’

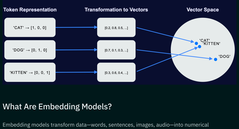
SEPTEMBER 13, 2024

Markku Räsänen

Embedding models transform data such as words, sentences, or images into numbers so that LLMs can understand their relationships—foundational to search, RAG, and generative AI.

In the last two years, there has been massive growth in generative AI, with market size projected to grow from **USD 20.9 billion in 2024 to USD 136.7 billion by 2030** at a compound annual growth rate of 36.7% according to MarketsandMarkets Research. This is one of the biggest disruptions in the digital age, enabling organizations to make sense of their data to lower operational costs and enhance decision-making.

One key piece behind this success is *embedding models*, which power large language models (LLMs). While LLMs and embedding models are related, they perform very different tasks. This post covers what embedding models are, how they work, and why they're essential building blocks of LLMs and many AI systems.



What Are Embedding Models?

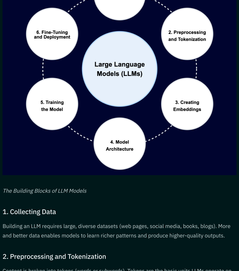
Embedding models transform data—words, sentences, images, audio—into numerical representations (vectors) that models can operate on. They embed items into a vector space based on relationships: closely related items are near each other; unrelated items are far apart. This allows AI systems to compute similarities and generate relevant content from user inputs (prompts).

What Is the Difference Between LLMs and Embedding Models?

LLMs don't "understand" language directly; they understand numerical relationships between tokens. Embeddings bridge that gap: they encode inputs into vectors LLMs can process.

Unlike LLMs, embedding models do not generate content; they encode data into vectors for comparison and analysis (often stored in a vector database). Many LLM systems include an embedding step for retrieval, ranking, or memory.

Essential Components of Large Language Models



The Building Blocks of LLM Models

1. Collecting Data

Building an LLM requires large, diverse datasets (web pages, social media, books, blogs). More and better data enables models to learn richer patterns and produce higher-quality outputs.

2. Preprocessing and Tokenization

Content is broken into tokens (words or subwords). Tokens are the basic units LLMs operate on.

3. Creating Embeddings

Embeddings transform tokens into vectors—numerical representations that capture meaning and relationships. Tokens with similar meanings (e.g., "cat" and "kitten") have similar vectors, enabling models to recognize semantic relatedness.

4. Model Architecture

Modern LLMs use transformer architectures introduced in the 2017 paper, "**Attention Is All You Need**." Transformers take token embeddings and apply attention mechanisms to model relationships in sequences, enabling generation of output text given input context.

5. Training the Model

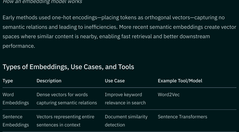
Models are trained to predict tokens and their relationships, adjusting internal parameters iteratively. This demands significant compute. After training, users can interact via natural language and receive outputs (text, images, audio).

6. Fine-Tuning and Deployment

Models can be fine-tuned for specific tasks. Once validated, they are deployed in applications like chatbots, virtual assistants, document analysis tools, and more.

Understanding Embeddings

Embedding models convert content into vectors so ML systems can reason about meaning and context.



How an embedding model works

Early methods used one-hot encodings—placing tokens as orthogonal vectors—capturing no semantic relations and leading to inefficiencies. More recent semantic embeddings create vector spaces where similar content is nearby, enabling fast retrieval and better downstream performance.

Types of Embeddings, Use Cases, and Tools

Type	Description	Use Case	Example Tool/Model
Word Embeddings	Dense vectors for words capturing semantic relations	Improve keyword relevance in search	Word2Vec
Sentence Embeddings	Vectors representing entire sentences in context	Document similarity detection	Sentence Transformers
Image Embeddings	Vectors capturing visual features	Image retrieval (find similar images)	TensorFlow Image Embedding API
Audio Embeddings	Vectors capturing salient audio features	Speech recognition	Whisper
Contextual Embeddings	Context-dependent vectors for the same token	Machine translation, NLU	BERT (Bidirectional Encoder Representations from Transformers)
Graph Embeddings	Vectors for nodes/graphs capturing relationships	Fraud detection via transaction patterns	Node2Vec
Multimodal Embeddings	Joint representations across text/images/audio	Enterprise AI integrating multiple inputs	CLIP

Classical vs. Semantic Approaches in Embeddings

As NLP has evolved, semantic approaches have transformed how machines represent and generate language.

Overview

Aspect	Classical Approaches	Semantic Approaches
Main Techniques	One-hot, Count-based, TF-IDF, N-grams	Word2Vec, Glove, ELMo
Context Consideration	Limited/none	Context-aware, semantic
Dimensionality	High-dimensional (e.g., vocab-size one-hot)	Lower-dimensional, dense
Semantic Capture	Limited	Strong
Long-distance Dependencies	Poor	Better (esp. with ELMo/transformers)
Training Method	Statistical/frequency	Neural predictive models
Scalability	Often more scalable (TF-IDF, count)	Less scalable; more complex training
Training Speed	Faster	Slower
Accuracy	Lower for complex NLP tasks	Higher, especially context-dependent
Flexibility	Fixed representations	Context-dependent, flexible
Handling Unseen Words	Poor	Better for rare/unseen
Applications	Basic NLP, IR	Sentiment, MT, QA, summarization, NLU

1) Classical Approach

- **Bag of Words (BoW):** Document vectors of word frequencies; ignores context, yields sparse vectors
- **TF-IDF:** Down-weights common terms, up-weights informative rare terms.
- **Co-occurrence Matrices:** Track proximity of words; still lacks semantic understanding.

Efficient for smaller datasets but weak on contextual meaning and generalization.

2) Semantic Approach (Embedding + LLM)

- **Word Embeddings (Word2Vec, GloVe):** Dense vectors place semantically similar words nearby.
- **Contextual Embeddings (BERT, GPT-family):** Vectors vary by surrounding context.
- **Fine-Tuning & Transfer:** Pretrained models can be adapted to many downstream tasks (QA, summarization, NLU, etc.).

This approach captures nuance in meaning, context, and syntax, improving accuracy and generalization across tasks.

Conclusion

Embeddings are foundational to LLMs, enabling them to "understand" human language or other data well enough to generate text, images, or audio. Without embeddings, we wouldn't have today's generative AI systems.

For businesses, leveraging embeddings effectively requires infrastructure and integration across open-source components—work that can take up to two years to build in-house. ConfidentialMind reduces this complexity by providing a platform to build secure AI systems and integrate AI into products and services.