Facsímil 01

Los cimientos

Capítulo 09

Del token al embedding: cómo el modelo representa lenguaje

686f6c61 · 2026

Capítulo 09: Del token al embedding: cómo el modelo representa lenguaje

Entrando en el tema

Escribes «Hola, ¿cómo estás?» y pulsas Enviar. Para ti son cuatro palabras, un signo de interrogación y una intención. Para el modelo, son números. Solo números.

Este capítulo explica el viaje que transforma tu texto en algo que una red neuronal puede procesar. Tiene dos etapas: la **tokenización** (trocear el texto en unidades procesables) y el **embedding** (convertir cada unidad en un vector de números que representa regularidades semánticas y de uso). Si el capítulo 4 fue el átomo (la neurona) y el capítulo 5 la molécula (la red), este capítulo es el sistema de coordenadas que permite operar con lenguaje sin que el modelo vea letras como las vemos nosotros.

Qué es un token

Un token es la **unidad mínima** que el modelo procesa. No es una palabra, aunque a veces coincide. No es un carácter, aunque a veces también. Es lo que el tokenizador (un programa que trocea texto) decide que es la unidad óptima para ese modelo.[1]

Algunos ejemplos con un tokenizador típico:

```
"Hola mundo"       → ["Hola", " mundo"]           (2 tokens)
"desarrolladores"  → ["des", "arrol", "ladores"]   (3 tokens)
"function get()"   → ["function", " get", "()"]   (3 tokens)
```

Observa tres cosas. Primero, los espacios importan: «mundo» y « mundo» son tokens distintos. El espacio indica que la palabra empieza tras otra. Segundo, las palabras largas o poco frecuentes se trocean en subtokens: «desarrolladores» se divide en tres piezas que el modelo puede recombinar. Tercero, el código se tokeniza de forma distinta al lenguaje natural: los paréntesis, llaves y operadores son tokens independientes.

El modelo no ve texto. Ve secuencias de números (IDs de token). Cada token tiene un identificador único en el vocabulario del modelo (típicamente entre 50 000 y 250 000 tokens distintos). Cuando escribes «Hola mundo», el modelo recibe algo como `[15496, 2159]`. Nunca ve las letras.

Todo se mide en tokens. La ventana de contexto (cuánto texto puede procesar el modelo de una vez), el precio de la API, el límite de respuesta: todo se mide en tokens. Es la moneda de la IA generativa.

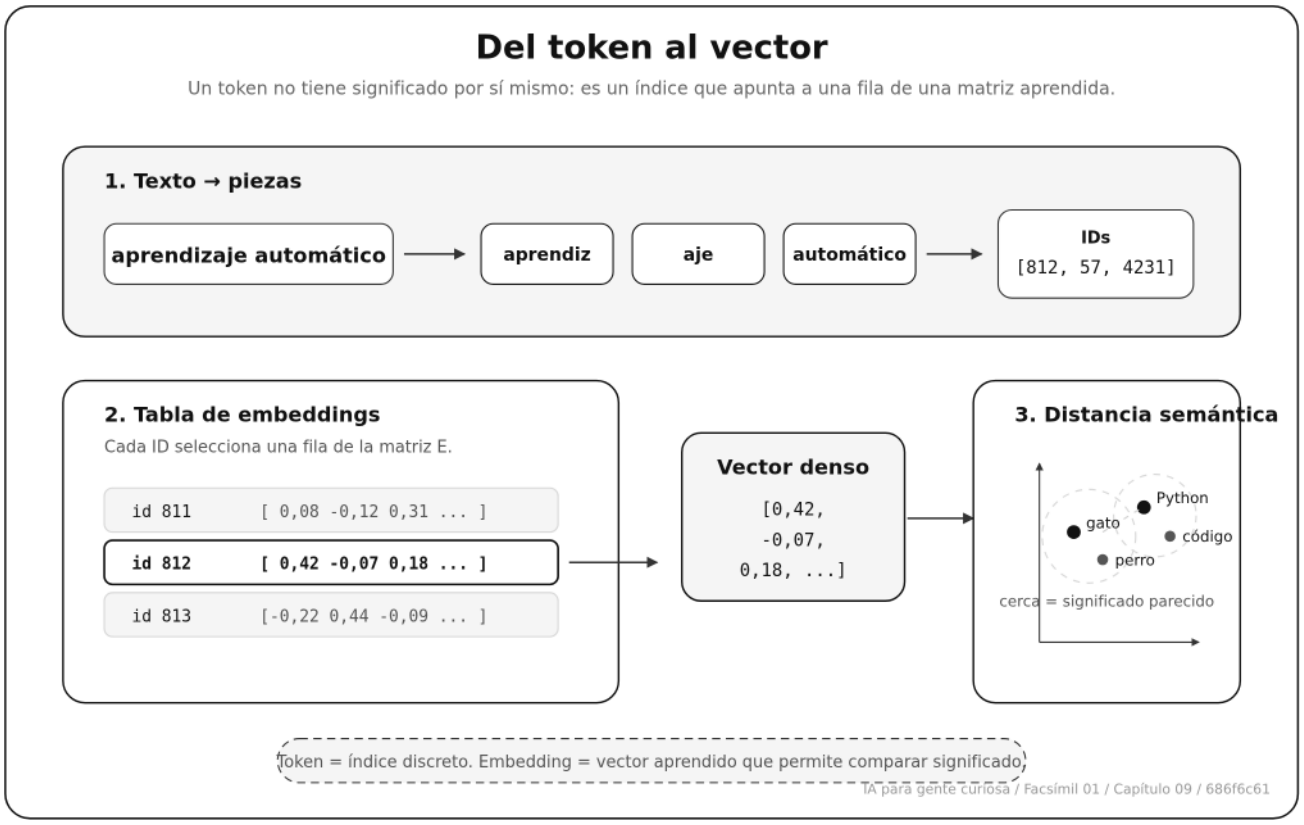
Cómo funciona la tokenización (BPE)

El algoritmo más usado se llama **Byte Pair Encoding** (BPE).[2] Funciona así:

1. Empieza con un vocabulario de caracteres individuales (a, b, c, ..., espacio, etc.).
2. Recorre todo el texto de entrenamiento y encuentra el **par de tokens consecutivos más frecuente**.
3. Fusiona ese par en un nuevo token y lo añade al vocabulario.
4. Repite miles de veces.

El resultado es un vocabulario donde las palabras muy frecuentes («el», «de», «y») son tokens únicos, mientras que las palabras raras o largas se descomponen en subtokens. «Desarrolladores» acaba siendo «des» + «arrol» + «ladores» porque esos fragmentos aparecen en muchas otras palabras («desarrollo», «arrollar», «desarrollado»). Es un equilibrio ingenioso: el vocabulario es manejable (50k-250k tokens) pero puede representar cualquier palabra.

El español gasta más tokens que el inglés. Como regla aproximada, un token equivale a 3-4 caracteres en texto latino. Pero el español, con sus tildes, sus conjugaciones verbales y sus palabras más largas, tiende a consumir más tokens que el inglés para expresar lo mismo. «Machine learning is great» son 4 tokens; «El aprendizaje automático es genial» son 7-8.



Qué es un embedding

Una vez que el texto se ha convertido en tokens, cada token necesita una representación numérica que la red pueda procesar. Esa representación es el **embedding**: un vector denso de números reales (típicamente entre 768 y 4096 dimensiones) que captura el significado del token en el contexto del modelo.[3]

«Denso» significa que casi todas las dimensiones tienen valores distintos de cero. Un vector *sparse* (disperso) tendría muchos ceros y unas pocas dimensiones activas. Un embedding es denso: cada dimensión contribuye un poco al significado global.

Las dimensiones no tienen etiquetas. No hay una dimensión que signifique «animalidad» y otra que signifique «tamaño». Cada dimensión es una coordenada en un espacio de alta dimensionalidad, y el significado emerge de la posición relativa del vector completo. Es como preguntar «¿qué significa la coordenada X = 0,23 en un mapa?». Nada por sí sola. Pero combinada con Y = -0,87 y las otras 1534 dimensiones, sitúa «gato» cerca de «felino» y lejos de «JavaScript».

```
"gato"       → [0.23, -0.87, 0.45, 0.12, ...]  (1536 dimensiones)
"felino"     → [0.21, -0.85, 0.48, 0.11, ...]  (muy cercano a "gato")
"JavaScript" → [-0.56, 0.33, -0.12, 0.78, ...] (completamente distinto)
```

Piensa en un mapa donde las palabras se colocan según su significado. «Perro» y «gato» están cerca (ambos son animales domésticos). «Python» y «JavaScript» están cerca (ambos son lenguajes de programación). Pero «perro» y «Python» están lejos. El embedding es la coordenada de cada palabra en ese mapa de miles de dimensiones.[4]

Embeddings en la práctica

Aritmética de significado

Una propiedad fascinante de los embeddings es que las relaciones semánticas a veces se expresan como operaciones vectoriales. Si el espacio de embeddings captura consistentemente las dimensiones de significado, las direcciones en ese espacio codifican relaciones:

```
rey - hombre + mujer ≈ reina
Madrid - España + Francia ≈ París
```

En notación vectorial, esto es literalmente sumar y restar los embeddings y buscar la palabra cuyo vector queda más cerca del resultado:

$$\mathbf{v}_{\text{rey}} - \mathbf{v}_{\text{hombre}} + \mathbf{v}_{\text{mujer}} \approx \mathbf{v}_{\text{reina}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\mathbf{v}_{\text{palabra}}$	El embedding (vector) de esa palabra	el vector de «rey»
$- \mathbf{v}_{\text{hombre}}$	Quitar la dirección «masculino»	restar el vector de «hombre»
$+ \mathbf{v}_{\text{mujer}}$	Añadir la dirección «femenino»	sumar el vector de «mujer»
$\approx$	El resultado cae cerca de	el vector de «reina»

En palabras: si a «rey» le restas «hombre» te queda algo parecido a la idea de «realeza», y al sumarle «mujer» llegas cerca de «reina». La dirección que separa «hombre» de «mujer» es parecida a la que separa «rey» de «reina».

Estas analogías, popularizadas por Word2Vec,[5] son ilustrativas, no garantizadas. Un embedding moderno no siempre produce estas operaciones limpiamente, pero el principio subyacente, que la dirección en el espacio codifica relaciones, es real y es la base de la búsqueda semántica.

Similitud semántica

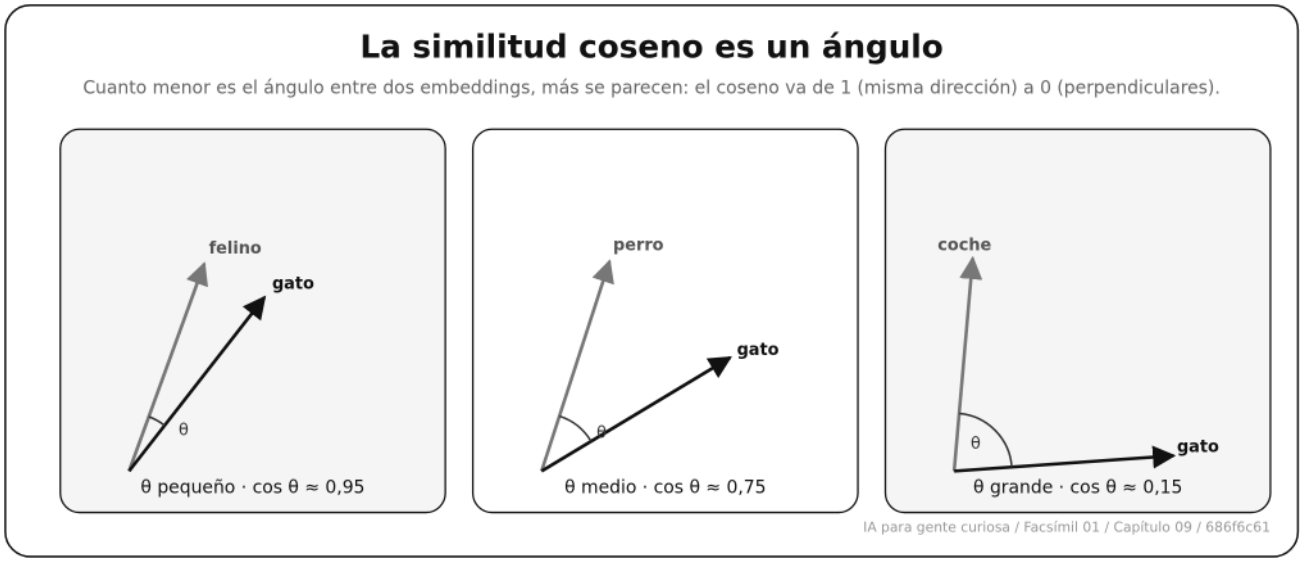
La medida estándar para comparar embeddings es la **similitud coseno**: el coseno del ángulo entre dos vectores. Vale 1 si apuntan exactamente en la misma dirección (idénticos), 0 si son perpendiculares (sin relación) y -1 si son opuestos. Su fórmula es el producto escalar de los dos vectores dividido por el producto de sus normas (sus longitudes):

$$\mathrm{sim}(\mathbf{a}, \mathbf{b}) = \cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|} = \frac{\sum_i a_i b_i}{\sum_i a_i^2 \quad \sum_i b_i^2}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
$\mathbf{a}$, $\mathbf{b}$	Los dos embeddings que comparas	el vector de «gato» y el de «felino»
$\mathbf{a} \cdot \mathbf{b}$	Producto escalar: suma de los productos componente a componente	$\sum_i a_i b_i$
$\|\mathbf{a}\|$	Norma (longitud) del vector, $\sum_i a_i^2$	cuánto «mide» el vector
a_i, b_i	La componente i de cada vector	la dimensión 23 de cada embedding
θ	Ángulo entre los dos vectores	pequeño si apuntan parecido

En palabras: el coseno mide si dos vectores apuntan en la misma dirección, sin importar su longitud. Por eso se prefiere a la distancia euclídea cuando solo interesa el significado y no la magnitud. Un detalle práctico: si los vectores están normalizados (norma 1), el denominador vale 1 y la similitud coseno se reduce al producto escalar, más barato de calcular. Por eso muchos índices vectoriales normalizan los embeddings de antemano.

PAR DE PALABRAS	SIMILITUD	INTERPRETACIÓN
«gato» / «felino»	Alta (~0.95)	Cuasi-sinónimos
«gato» / «perro»	Media (~0.75)	Misma categoría
«gato» / «coche»	Baja (~0.15)	Sin relación
«deploy» / «desplegar»	Alta (~0.90)	Traducción (modelo multilingüe)



La clave: conceptos similares quedan **cerca** en el espacio vectorial. Esto permite buscar por significado, no por palabras exactas. «¿Cómo despliego en producción?» encuentra documentos sobre «deploy to prod» aunque no compartan ni una sola palabra.

Modelos de embedding (2026)

Fecha de corte: 10 de junio de 2026. Esta tabla no es un ranking universal: es una foto de familias útiles para entender decisiones de ingeniería. Antes de elegir proveedor hay que comprobar licencia, coste, límites, idioma, evaluación propia y si el índice vectorial ya existente obliga a re-embedar documentos.

MODELO	PROVEEDOR	DIMENSIONES DECLARADAS	QUÉ APORTA REALMENTE
`text-embedding-3-large`	OpenAI	3072 por defecto; reducible con `dimensions`	API madura y buen punto de partida general. Reducir dimensiones ahorra almacenamiento, pero hay que medir recuperación en tus datos.[6]
`voyage-4-large`	Voyage AI	1024 por defecto; 256, 512, 1024 o 2048	Familia orientada a recuperación y RAG multilingüe con dimensiones Matryoshka y compatibilidad dentro de la serie Voyage 4.[7]
`Qwen3-Embedding-8B`	Qwen (open weights)	hasta 4096; salida configurable de 32 a 4096	Opción autoalojable para equipos que necesitan control de pesos, idiomas y despliegue propio, a cambio de gestionar hardware y rendimiento.[8]
`BGE-M3`	BAAI (open weights)	1024	Modelo abierto multilingüe útil para búsqueda híbrida (densa, dispersa y multi-vector) y entornos donde se quiere controlar el índice y el runtime.[9] Conviene evaluarlo frente al dominio concreto antes de asumir que “open” significa suficiente.
`embed-v4`	Cohere	1536 por defecto; 256, 512, 1024 o 1536	Embeddings multimodales de texto e imagen, útiles para documentos visuales, PDFs, catálogos o búsqueda mixta.[10]

Importante: el modelo de embedding convierte entradas en vectores para comparar, ordenar o recuperar. No genera texto. Es distinto del LLM generativo. En un RAG típico, el embedding sirve para **buscar** fragmentos relevantes; el LLM sirve para **redactar** una respuesta usando esos fragmentos. Confundir ambas piezas lleva a arquitecturas caras y difíciles de depurar.

Cómo se entrenan los embeddings

El objetivo de entrenamiento es simple y brillante: dadas dos palabras que aparecen cerca en un texto, sus embeddings deben ser similares. Dadas dos palabras que no aparecen juntas, sus embeddings deben ser distintos.[11] El modelo aprende de millones de frases: si «gato» y «felino» aparecen en contextos similares («el _ _ _ _ duerme», «acaricié al _ _ _ _»), sus vectores se acercan. Si «gato» y «tornillo» nunca comparten contexto, sus vectores se alejan.

Este principio, «dime con qué palabras apareces y te diré qué significas», se llama **hipótesis distribucional** y es la base de casi todos los embeddings modernos. No hace falta que nadie etiquete datos: el propio texto proporciona la señal de aprendizaje. Es aprendizaje auto-supervisado a escala masiva.

Embeddings Matryoshka: menos dimensiones sin perder calidad

Una innovación reciente son los **embeddings Matryoshka** (MRL, *Matryoshka Representation Learning*).[1][2] La idea: entrenar el embedding para que funcione bien **a cualquier dimensionalidad**. Puedes tomar las primeras 256 dimensiones de un embedding de 1024 y obtener una representación de calidad razonable, o usar las 1024 completas para máxima precisión. Es como tener varios embeddings de distinto tamaño dentro del mismo vector, como las muñecas rusas que le dan nombre.

Esto resuelve un problema práctico enorme: no necesitas elegir entre precisión y coste de antemano. Usas 256 dimensiones para búsquedas rápidas y baratas, y 1024 para cuando necesitas la máxima calidad. Modelos como voyage-4-large, Qwen3-Embedding y Cohere embed-v4 ya lo implementan.

En el día a día

Búsqueda semántica y RAG. Es el uso más extendido y el que mejor enseña la cadena completa. Primero troceas tus documentos en fragmentos manejables porque el contexto del LLM es limitado y un párrafo enfocado recupera mejor que un manual entero. Pasas cada fragmento por el modelo de embedding y guardas el vector resultante junto al texto original en una base de datos vectorial como Pinecone, Chroma o pgvector. Cuando llega una pregunta, conviertes esa pregunta al mismo espacio vectorial con el mismo modelo y pides al índice los fragmentos cuya similitud coseno es mayor. Esos fragmentos se inyectan en el prompt para que el LLM redacte una respuesta apoyada en tu documentación y no en lo que recuerde de su entrenamiento. La decisión de ingeniería crítica es que la consulta y los documentos deben compartir modelo y normalización: si reindexas con otro modelo, tienes que recalcular todos los vectores. Conviene afinar el tamaño del fragmento, el número de resultados que recuperas y, si la precisión no basta, añadir un reordenador que repase los candidatos.

Clasificación. Puedes agrupar tickets de soporte, correos o reseñas por similitud semántica sin redactar reglas ni definir categorías a mano. Calculas el embedding de cada texto y agrupas los vectores cercanos, o comparas cada entrada nueva con vectores de ejemplo ya etiquetados y le asignas la etiqueta del más parecido. La pieza que interviene es siempre la misma medida de distancia en el espacio vectorial, pero la decisión está en elegir un modelo de embedding alineado con tu dominio e idioma, porque un modelo entrenado para inglés general clasificará mal jerga médica en español. Lo que hay que cuidar es validar con datos reales dónde poner el umbral que separa una categoría de otra, ya que ese corte determina cuántos casos quedan mal asignados.

Detección de duplicados. Comparas embeddings para encontrar textos que dicen lo mismo con palabras distintas, algo que una comparación literal o un hash nunca detectarían. Sirve para deduplicar una base de conocimiento, detectar reseñas falsas repetidas o evitar incidencias que ya existen. La mecánica es calcular la similitud coseno entre pares de vectores y marcar como sospechosos los que superan un umbral alto. Lo delicado es fijar ese

umbral con cuidado, porque demasiado bajo fusiona textos parecidos pero distintos y demasiado alto deja pasar duplicados reescritos. A escala grande no comparas todos los pares uno a uno sino que usas el índice vectorial para traer solo los vecinos más cercanos de cada texto.

Traducción y multilingüe. Los modelos multilingües colocan «deploy» cerca de «desplegar» y «perro» cerca de «dog» en el mismo espacio vectorial, porque aprendieron a representar el significado por encima del idioma. Esto permite buscar en español sobre una base de documentos en inglés sin traducir nada de antemano: la consulta y los documentos caen cerca si comparten significado aunque no compartan ni una palabra. La decisión de ingeniería es elegir un modelo entrenado de verdad para varios idiomas y no uno monolingüe, y comprobar que rinde en el par de idiomas que te importa. Hay que cuidar que la calidad del alineamiento varía según los idiomas, así que conviene evaluar con consultas reales antes de confiar en que la búsqueda cruzada funciona en tu caso.

Por qué debería importarte

Los embeddings son el pegamento que conecta el mundo del texto con el mundo de las matemáticas. Sin ellos, una red neuronal no podría procesar lenguaje: solo sabe multiplicar matrices de números.

Cada vez que usas un LLM, hay embeddings trabajando: los tokens de entrada se convierten en embeddings antes de entrar al Transformer. Cada vez que usas búsqueda semántica, hay embeddings trabajando: la consulta y los documentos se comparan en el espacio vectorial. Son ubicuos y silenciosos. Entenderlos te permite elegir el modelo adecuado, ajustar la dimensionalidad según tu presupuesto y depurar por qué tu búsqueda semántica devuelve resultados incoherentes.

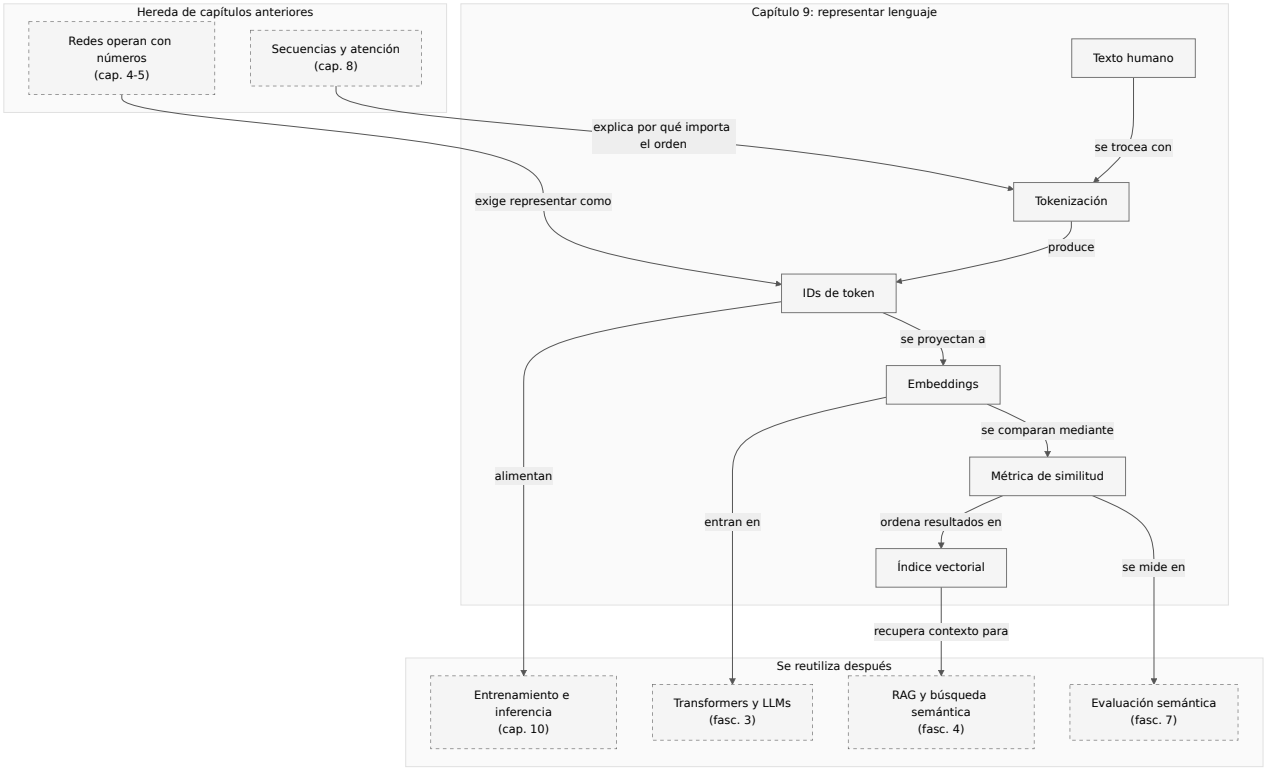
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Confundir token con palabra	«Hola mundo» son dos palabras, pero con BPE pueden ser 2-3 tokens. «Desarrolladores» es una palabra pero 3-4 tokens. Planificar costes en palabras te dará sorpresas en la factura.	Mide en tokens, no en palabras. Usa el tokenizador del proveedor para contar.
Tratar la métrica vectorial como una verdad universal	Coseno, producto escalar y distancia euclídea pueden comportarse distinto según normalización, modelo e índice. Decir “euclídea nunca sirve” es tan pobre como usarla sin pensar.	Lee la recomendación del modelo y de la base vectorial. Si los vectores están normalizados, coseno y producto escalar pueden ser equivalentes. Evalúa la métrica con consultas reales.
Usar el mismo modelo de embedding para todo	Un modelo optimizado para inglés puede rendir mal en español jurídico. Un modelo de código no sirve para reseñas de restaurantes.	Elige el modelo de embedding según tu dominio, idioma y tipo de contenido.
Asumir que más dimensiones siempre es mejor	Más dimensiones permiten más matices, pero también más memoria, más latencia y más coste de almacenamiento.	Evalúa con tus datos. A veces 256 dimensiones bastan; a veces necesitas 3072.

Cómo encaja todo

Este mapa se lee como una tubería de representación. Venimos de neuronas, capas y arquitecturas que solo operan con números; este capítulo explica cómo el texto entra en ese mundo numérico sin fingir que el modelo “lee” como una persona. La decisión técnica nueva es doble: elegir cómo troceas el texto y elegir qué espacio vectorial usarás para comparar significado.

La consecuencia aparece después en casi todo el facsímil. Entrenar e inferir necesitan tokens; RAG necesita embeddings y métricas de similitud; evaluar sistemas semánticos exige saber cuándo una búsqueda recupera por significado y cuándo solo recupera ruido con buena pinta.


Hereda de capítulos anteriores
Redes operan con números (cap. 4-5)
Secuencias y atención (cap. 8)
Capítulo 9: representar lenguaje
Texto humano
explica por qué importa el orden
se trocea con
exige representar como
Tokenización
produce
IDs de token
se proyectan a
Embeddings
se comparan mediante
alimentan
Métrica de similitud
entran en
ordena resultados en
Índice vectorial
se mide en
recupera contexto para
Se reutiliza después
Entrenamiento e inferencia (cap. 10)
Transformers y LLMs (fasc. 3)
RAG y búsqueda semántica (fasc. 4)
Evaluación semántica (fasc. 7)

Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Token	Unidad mínima de texto procesada por el modelo. Puede ser palabra, subtoken o carácter.
Tokenizador	Programa que convierte texto en secuencias de tokens según un vocabulario predefinido.
BPE (Byte Pair Encoding)	Algoritmo de tokenización que parte de caracteres y fusiona los pares más frecuentes hasta formar un vocabulario de subpalabras.
Subtoken	Fragmento de palabra en que se descompone un término largo o poco frecuente para poder representarlo con piezas reutilizables.
Embedding	Vector denso de números reales que representa un token, frase o documento en un espacio semántico de alta dimensionalidad.
Similitud coseno	Medida de similitud entre vectores basada en el coseno de su ángulo. Es muy habitual en embeddings, pero debe validarse con la recomendación del modelo y del índice.
Producto escalar	Suma de los productos componente a componente de dos vectores, numerador de la similitud coseno.
Norma	Longitud de un vector, calculada como la raíz de la suma de sus componentes al cuadrado.
Hipótesis distribucional	Idea de que el significado de una palabra se deduce de las palabras con las que suele aparecer.
Aprendizaje auto-supervisado	Entrenamiento en el que la señal de aprendizaje sale del propio texto, sin necesidad de datos etiquetados a mano.
Embeddings Matryoshka	Embeddings entrenados para funcionar bien a varias dimensionalidades, permitiendo recortar el vector para equilibrar precisión y coste.
Base de datos vectorial	Sistema de almacenamiento optimizado para buscar vectores cercanos por similitud.

Antes de pasar página

- [ ] ¿Puedo explicar la diferencia entre token y palabra con un ejemplo? (Si no, vuelve a «Qué es un token».)
- [ ] ¿Entiendo qué es un embedding y por qué las dimensiones no tienen etiquetas? (Si no, vuelve a «Qué es un embedding».)

- [ ] ¿Sé qué mide la similitud coseno? (Si no, vuelve a «Similitud semántica».)
- [ ] ¿Puedo nombrar al menos tres usos prácticos de los embeddings? (Si no, vuelve a «En el día a día».)
- [ ] ¿Sé explicar por qué dos textos parecidos pueden tener tokens distintos? (Si no, vuelve a «Qué es un token».)

En resumen

IDEA FUERZA	DETALLE
Un token es la unidad mínima de texto para el modelo.	No es una palabra: «desarrolladores» son 3 tokens. Todo se mide en tokens: contexto, precio, límites.
Un embedding es un vector denso que representa regularidades semánticas y de uso.	Cientos o miles de números sitúan cada concepto en un espacio vectorial. Conceptos usados de forma parecida tienden a quedar cerca; conceptos distintos, lejos.
Los embeddings son el pegamento entre texto y matemáticas.	Sin ellos, las redes neuronales no podrían procesar lenguaje. Con ellos, puedes buscar por significado, no por palabras exactas.

Para saber más

Bengio, Y., Ducharme, R., Vincent, P. y Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 1137-1155. https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D. y Liu, Z. (2024). BGE M3-Embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv:2402.03216. https://arxiv.org/abs/2402.03216

Cohere. (2025). *Announcing Embed Multimodal v4*. https://docs.cohere.com/changelog/embed-multimodal-v4

Devlin, J., Chang, M. W., Lee, K. y Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. En *Proceedings of NAACL-HLT* (pp. 4171-4186). https://doi.org/10.18653/v1/N19-1423

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P. y Farhadi, A. (2022). Matryoshka representation learning. En *Advances in Neural Information Processing Systems 35*. https://arxiv.org/abs/2205.13147

Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781

OpenAI. (2026). *Vector embeddings*. https://developers.openai.com/api/docs/guides/embeddings

Qwen. (2025). *Qwen3-Embedding-8B*. Hugging Face. https://huggingface.co/Qwen/Qwen3-Embedding-8B

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Sennrich, R., Haddow, B. y Birch, A. (2016). Neural machine translation of rare words with subword units. En *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (pp. 1715-1725). https://doi.org/10.18653/v1/P16-1162

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. y Polosukhin, I. (2017). Attention is all you need. En *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). https://papers.nips.cc/paper/7181-attention-is-all-you-need

Voyage AI. (2026). *Text embeddings*. https://docs.voyageai.com/docs/embeddings

---

Notas

1. Brown, T. B. y otros (2020). Language models are few-shot learners. En *Advances in Neural Information Processing Systems 33* (pp. 1877-1901). https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html. GPT-3 usó tokenización por pares de bytes (BPE), que equilibra el tamaño del vocabulario con la capacidad de representar palabras raras como secuencias de subtokens. ↩

2. Sennrich, R., Haddow, B. y Birch, A. (2016). Neural machine translation of rare words with subword units. En *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (pp. 1715-1725). https://doi.org/10.18653/v1/P16-1162. BPE se popularizó como método de tokenización por subpalabras, resolviendo el problema de las palabras fuera de vocabulario en traducción automática y siendo adoptado posteriormente por GPT y la mayoría de LLMs. ↩

3. Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781. Word2Vec demostró que las representaciones vectoriales densas de palabras capturan relaciones semánticas, permitiendo operaciones como «rey - hombre + mujer ≈ reina». ↩

4. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El capítulo 12 aborda las representaciones distribuidas y cómo los embeddings densos aprendidos permiten a los modelos capturar relaciones semánticas complejas. ↩

5. Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781. Las analogías semánticas vectoriales fueron una de las demostraciones más impactantes de Word2Vec, mostrando que los embeddings capturan relaciones sistemáticas entre conceptos. ↩

6. OpenAI. (2026). *Vector embeddings*. https://developers.openai.com/api/docs/guides/embeddings. La documentación indica 1536 dimensiones para `text-embedding-3-small`, 3072 para `text-embedding-3-large` y soporte para reducir dimensiones con el parámetro `dimensions`. ↩

7. Voyage AI. (2026). *Text embeddings*. https://docs.voyageai.com/docs/embeddings. La documentación de Voyage 4 declara contexto de 32 000 tokens y dimensiones 256, 512, 1024 y 2048 para `voyage-4-large`. ↩

8. Qwen. (2025). *Qwen3-Embedding-8B*. Hugging Face. https://huggingface.co/Qwen/Qwen3-Embedding-8B. La ficha declara 100+ idiomas, contexto de 32k y dimensiones configurables hasta 4096. ↩

9. Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D. y Liu, Z. (2024). BGE M3-Embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. arXiv:2402.03216. https://arxiv.org/abs/2402.03216. El artículo presenta BGE-M3 como modelo multilingüe, multifuncional y de granularidad variable, con entradas hasta 8192 tokens. ↩

10. Cohere. (2025). *Announcing Embed Multimodal v4*. https://docs.cohere.com/changelog/embed-multimodal-v4. Cohere declara dimensiones Matryoshka 256, 512, 1024 y 1536, contexto de 128k y soporte multimodal. ↩

11. Mikolov, T., Chen, K., Corrado, G. y Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781. Word2Vec popularizó dos arquitecturas: skip-gram (predecir contexto a partir de palabra) y CBOW (predecir palabra a partir de contexto), demostrando que ambas producen embeddings de alta calidad. ↩

12. Kusupati, A., Bhatt, G., Rege, A., Wallingford, M., Sinha, A., Ramanujan, V., Howard-Snyder, W., Chen, K., Kakade, S., Jain, P. y Farhadi, A. (2022). Matryoshka representation learning. En *Advances in Neural Information Processing Systems 35*. https://arxiv.org/abs/2205.13147. MRL

entrena embeddings para que sean útiles a múltiples dimensionalidades, permitiendo elegir entre precisión y coste sin reentrenar. ↩