

In [ ]:

# Exploración de Embeddings: Modelos Livianos para Retrieval

Comparamos modelos de sentence-transformers enfocándonos en:

- **Tamaño y velocidad** — cuál es el más liviano
- **Calidad de retrieval** — precision@k, recall@k, MRR
- **Trade-off práctico** — el mejor balance peso/rendimiento

## 1. Setup e imports

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sentence_transformers import SentenceTransformer
import faiss
import time
import os

plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)

# Modelos a comparar: nombre -> (id huggingface, descripcion, prefijo_query,
# None = sin prefijo. Los modelos e5 y nomic necesitan prefijos especiales.
MODELS = {
    'MiniLM-L6': ('all-MiniLM-L6-v2', '22M -
    'MiniLM-L12': ('all-MiniLM-L12-v2', '33M -
    'paraphrase-multi-L12': ('paraphrase-multilingual-MiniLM-L12-v2', '118M -
    'multi-e5-small': ('intfloat/multilingual-e5-small', '118M -
    'nomic-v1.5': ('nomic-ai/nomic-embed-text-v1.5', '137M -
    'BGE-m3': ('BAAI/bge-m3', '568M -
}

print(f'Modelos a comparar: {len(MODELS)}')
for name, (model_id, desc, _, _) in MODELS.items():
    print(f' {name:25s} → {model_id:50s} ({desc})')
```

Modelos a comparar: 6

MiniLM-L6	→ all-MiniLM-L6-v2
(22M - solo inglés (baseline))	
MiniLM-L12	→ all-MiniLM-L12-v2
(33M - solo inglés)	
paraphrase-multi-L12	→ paraphrase-multilingual-MiniLM-L12-v2
(118M - multilingüe 50+ idiomas)	
multi-e5-small	→ intfloat/multilingual-e5-small
(118M - multilingüe, buen retrieval)	
nomic-v1.5	→ nomic-ai/nomic-embed-text-v1.5
(137M - Matryoshka, dim flexible)	
BGE-m3	→ BAAI/bge-m3
(568M - multilingüe SOTA, más pesado)	

## 2. Corpus de prueba

Creamos un corpus con documentos de distintas categorías para evaluar retrieval. Cada query tiene documentos relevantes conocidos (ground truth).

```
In [2]: # Corpus: documentos agrupados por tema (en español para evaluar retrieval)
corpus = [
    # Programación (0-4)
    'Python es un lenguaje de programación de alto nivel conocido por su leg',
    'JavaScript se ejecuta en el navegador y es esencial para el desarrollo',
    'Rust proporciona seguridad de memoria sin recolector de basura mediante',
    'SQL es el lenguaje estándar para consultar bases de datos relacionales',
    'Git es un sistema de control de versiones distribuido para rastrear cam

    # Ciencia (5-9)
    'La fotosíntesis convierte la luz solar en energía química dentro de las',
    'La teoría de la relatividad describe cómo la gravedad deforma el espaci',
    'El ADN transporta información genética usando cuatro bases de nucleótic',
    'El entrelazamiento cuántico vincula partículas sin importar la distanci',
    'La evolución ocurre mediante selección natural y deriva genética',

    # Cocina (10-14)
    'El pan de masa madre requiere un cultivo fermentado de harina y agua',
    'El arroz para sushi se sazona con vinagre, azúcar y sal',
    'La reacción de Maillard crea el dorado y sabor al cocinar carne a alta',
    'La emulsificación une aceite y agua usando agentes como la yema de huer',
    'La fermentación preserva alimentos y crea sabores complejos con el tien

    # Finanzas (15-19)
    'El interés compuesto hace crecer los ahorros exponencialmente a largo p',
    'La diversificación reduce el riesgo del portafolio distribuyendo invers',
    'La inflación erosiona el poder adquisitivo y afecta las inversiones de',
    'La Reserva Federal establece política monetaria ajustando las tasas de',
    'Los fondos indexados replican índices de mercado con comisiones mínimas

    # Geografía (20-24)
    'La selva amazónica produce aproximadamente el 20% del oxígeno del mund',
    'Las placas tectónicas se desplazan causando terremotos y erupciones vol',
    'El desierto del Sahara se extiende por once países del norte de África',
    'Las corrientes oceánicas regulan el clima global y los patrones meteorol
```

```

    'La Fosa de las Marianas es el punto más profundo conocido del océano',
]

# Queries en español con ground truth (índices de documentos relevantes)
queries = [
    ('¿Cómo manejan los lenguajes de programación la memoria?', [6]),
    ('¿Qué es el control de versiones para código fuente?', [4]),
    ('¿Cómo producen energía las plantas a partir de la luz?', [5]),
    ('Cuéntame sobre física de partículas y mecánica cuántica', [7]),
    ('¿Cómo funciona la fermentación del pan?', [1]),
    ('¿Qué hace que la comida sepa mejor al cocinarla a fuego alto?', [12]),
    ('¿Cómo puedo hacer crecer mis ahorros con el tiempo?', [15]),
    ('¿Qué afecta la economía y los precios?', [17]),
    ('Háblame de las partes más profundas del océano', [2]),
    ('¿Cómo cambia la superficie de la Tierra con el tiempo?', [21])
]

print(f'Corpus: {len(corpus)} documentos en español')
print(f'Queries: {len(queries)} en español con ground truth')
print(f'Categorías: programación, ciencia, cocina, finanzas, geografía')

```

Corpus: 25 documentos en español

Queries: 10 en español con ground truth

Categorías: programación, ciencia, cocina, finanzas, geografía

### 3. Cargar modelos y generar embeddings

Cargamos cada modelo, medimos tiempo de carga, tiempo de encoding, y tamaño en memoria.

In [3]: **import** gc

```

def measure_model(name, model_id, corpus, queries, q_prefix=None, d_prefix=None):
    """Carga modelo, genera embeddings, mide tiempos y tamaño."""
    gc.collect()

    # Cargar modelo (nomic necesita trust_remote_code)
    t0 = time.perf_counter()
    kwargs = {'trust_remote_code': True} if 'nomic' in model_id else {}
    model = SentenceTransformer(model_id, **kwargs)
    load_time = time.perf_counter() - t0

    # Tamaño del modelo en MB (acceder al transformer base)
    try:
        base = model[0].auto_model
    except:
        base = model
    param_size = sum(p.nelement() * p.element_size() for p in base.parameters())
    model_mb = param_size / (1024 * 1024)
    n_params = sum(p.nelement() for p in base.parameters()) / 1e6

    # Aplicar prefijos si el modelo los requiere
    corpus_input = [f'{d_prefix}{doc}' for doc in corpus] if d_prefix else corpus
    query_texts = [f'{q_prefix}{q}' for q, _ in queries] if q_prefix else queries

```

```

# Encoding del corpus
t0 = time.perf_counter()
corpus_embs = model.encode(corpus_input, normalize_embeddings=True, show_p
corpus_time = time.perf_counter() - t0

# Encoding de queries
t0 = time.perf_counter()
query_embs = model.encode(query_texts, normalize_embeddings=True, show_p
query_time = time.perf_counter() - t0

dim = corpus_embs.shape[1]

print(f'{name:25s} | dim={dim:4d} | {n_params:.1f}M params | {model_mb:6
      f'load={load_time:5.1f}s | corpus={corpus_time:.3f}s | queries={qu

return {
    'name': name,
    'model_id': model_id,
    'dim': dim,
    'n_params_m': round(n_params, 1),
    'model_mb': round(model_mb, 1),
    'load_time_s': round(load_time, 2),
    'corpus_encode_s': round(corpus_time, 4),
    'query_encode_s': round(query_time, 4),
    'corpus_embs': corpus_embs,
    'query_embs': query_embs,
}

# Ejecutar benchmark de carga y encoding
results = {}
print(f'{"Modelo":25s} | {"Dim":>4s} | {"Params":>10s} | {"Size":>6s} | {"L
print('-' * 115)
for name, (model_id, desc, q_prefix, d_prefix) in MODELS.items():
    results[name] = measure_model(name, model_id, corpus, queries, q_prefix,

```

Modelo	Dim	Params	Size	Load	Corpus
Queries					

Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF\_TOKEN to enable higher rate limits and faster downloads.
Loading weights: 0%| | 0/103 [00:00<?, ?it/s]
BertModel LOAD REPORT from: sentence-transformers/all-MiniLM-L6-v2
Key | Status | |
-----+-----+---
embeddings.position\_ids | UNEXPECTED | |
Notes:
- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.
MiniLM-L6 | dim= 384 | 22.7M params | 87MB | load= 2.4s
| corpus=0.264s | queries=0.015s
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]

**BertModel LOAD REPORT** from: sentence-transformers/all-MiniLM-L12-v2

Key	Status		
-----+-----+--			
embeddings.position_ids	UNEXPECTED		

## Notes:

- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.

MiniLM-L12 | dim= 384 | 33.4M params | 127MB | load= 1.9s  
| corpus=0.018s | queries=0.012s  
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]

**BertModel LOAD REPORT** from: sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2

Key	Status		
-----+-----+--			
embeddings.position_ids	UNEXPECTED		

## Notes:

- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.

paraphrase-multi-L12 | dim= 384 | 117.7M params | 449MB | load= 3.5s  
| corpus=0.041s | queries=0.014s  
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]

**BertModel LOAD REPORT** from: intfloat/multilingual-e5-small

Key	Status		
-----+-----+--			
embeddings.position_ids	UNEXPECTED		

## Notes:

- UNEXPECTED: can be ignored when loading from different task/architecture; not ok if you expect identical arch.

multi-e5-small | dim= 384 | 117.7M params | 449MB | load= 4.1s  
| corpus=0.055s | queries=0.017s  
model.safetensors: 0%| | 0.00/547M [00:00<?, ?B/s]

## &lt;All keys matched successfully&gt;

tokenizer\_config.json: 0.00B [00:00, ?B/s]

vocab.txt: 0.00B [00:00, ?B/s]

tokenizer.json: 0.00B [00:00, ?B/s]

special\_tokens\_map.json: 0%| | 0.00/695 [00:00<?, ?B/s]

config.json: 0%| | 0.00/286 [00:00<?, ?B/s]

nomic-v1.5 | dim= 768 | 136.7M params | 522MB | load= 19.0s  
| corpus=0.055s | queries=0.018s

modules.json: 0%| | 0.00/349 [00:00<?, ?B/s]

config\_sentence\_transformers.json: 0%| | 0.00/123 [00:00<?, ?B/s]

README.md: 0.00B [00:00, ?B/s]

sentence\_bert\_config.json: 0%| | 0.00/54.0 [00:00<?, ?B/s]

config.json: 0%| | 0.00/687 [00:00<?, ?B/s]

pytorch\_model.bin: 0%| | 0.00/2.27G [00:00<?, ?B/s]

Loading weights: 0%| | 0/391 [00:00<?, ?it/s]

tokenizer\_config.json: 0%| | 0.00/444 [00:00<?, ?B/s]

sentencepiece.bpe.model: 0%| | 0.00/5.07M [00:00<?, ?B/s]

model.safetensors: 0%| | 0.00/2.27G [00:00<?, ?B/s]

tokenizer.json: 0%| | 0.00/17.1M [00:00<?, ?B/s]

special\_tokens\_map.json: 0%| | 0.00/964 [00:00<?, ?B/s]

config.json: 0%| | 0.00/191 [00:00<?, ?B/s]

BGE-m3 | dim=1024 | 567.8M params | 2166MB | load= 48.0s | corpus=0.069s | queries=0.029s

## 4. Evaluación de Retrieval

Para cada modelo, usamos FAISS para buscar los k documentos más cercanos a cada query y comparamos contra el ground truth.

### Métricas:

- **Precision@k**: de los k recuperados, qué fracción es relevante
- **Recall@k**: de los relevantes, qué fracción fue recuperada
- **MRR (Mean Reciprocal Rank)**: inverso de la posición del primer resultado relevante

```
In [4]: def evaluate_retrieval(corpus_embs, query_embs, queries, k=5):
        """Evalúa retrieval con FAISS. Retorna métricas por query y agregadas."""
        dim = corpus_embs.shape[1]

        # Índice FAISS con inner product (embeddings ya normalizados = cosine si
        index = faiss.IndexFlatIP(dim)
        index.add(corpus_embs.astype(np.float32))

        # Buscar k vecinos para cada query
        scores, indices = index.search(query_embs.astype(np.float32), k)

        per_query = []
        for i, (query_text, relevant_ids) in enumerate(queries):
            retrieved = indices[i].tolist()
            relevant_set = set(relevant_ids)
            retrieved_relevant = [idx for idx in retrieved if idx in relevant_set]

            precision_at_k = len(retrieved_relevant) / k
            recall_at_k = len(retrieved_relevant) / len(relevant_set) if relevant_set

            # MRR: posición del primer relevante
            rr = 0
            for rank, idx in enumerate(retrieved, 1):
                if idx in relevant_set:
                    rr = 1.0 / rank
                    break

            per_query.append({
                'query': query_text[:60],
                'retrieved': retrieved,
                'relevant': relevant_ids,
                'precision@k': precision_at_k,
                'recall@k': recall_at_k,
                'mrr': rr,
            })

        return {
            'per_query': per_query,
```

```

        'mean_precision': np.mean([q['precision@k'] for q in per_query]),
        'mean_recall': np.mean([q['recall@k'] for q in per_query]),
        'mrr': np.mean([q['mrr'] for q in per_query]),
    }

    # Evaluar cada modelo
    K = 5
    eval_results = {}
    for name, res in results.items():
        ev = evaluate_retrieval(res['corpus_embs'], res['query_embs'], queries,
                               eval_results[name] = ev
        print(f"{name:15s} | Precision@{K}={ev['mean_precision']:.3f} | Recall@{K}={ev['mean_recall']:.3f} | MRR={ev['mrr']:.3f}")

```

MiniLM-L6	Precision@5=0.240	Recall@5=0.733	MRR=0.778
MiniLM-L12	Precision@5=0.220	Recall@5=0.683	MRR=0.820
paraphrase-multi-L12	Precision@5=0.320	Recall@5=0.900	MRR=0.933
multi-e5-small	Precision@5=0.300	Recall@5=0.867	MRR=0.950
nomiic-v1.5	Precision@5=0.240	Recall@5=0.733	MRR=0.900
BGE-m3	Precision@5=0.300	Recall@5=0.867	MRR=1.000

## 5. Tabla comparativa resumen

```

In [5]: # Tabla resumen: velocidad + retrieval
rows = []
for name in results:
    r = results[name]
    ev = eval_results[name]
    rows.append({
        'Modelo': name,
        'Params (M)': r['n_params_m'],
        'Tamaño (MB)': r['model_mb'],
        'Dim': r['dim'],
        'Load (s)': r['load_time_s'],
        'Encode corpus (s)': r['corpus_encode_s'],
        'Encode queries (s)': r['query_encode_s'],
        'Precision@5': round(ev['mean_precision'], 3),
        'Recall@5': round(ev['mean_recall'], 3),
        'MRR': round(ev['mrr'], 3),
    })

df_summary = pd.DataFrame(rows).set_index('Modelo')
df_summary.style.background_gradient(subset=['Precision@5', 'Recall@5', 'MRR'],

```

Out[5]:

	Params (M)	Tamaño (MB)	Dim	Load (s)	Encode corpus (s)	Encode queries (s)	Pr
Modelo							
MiniLM-L6	22.700000	86.600000	384	2.360000	0.264300	0.014600	
MiniLM-L12	33.400000	127.300000	384	1.850000	0.017600	0.012400	
paraphrase- multi-L12	117.700000	448.800000	384	3.500000	0.040700	0.013800	
multi-e5- small	117.700000	448.800000	384	4.100000	0.055500	0.016900	
nomic-v1.5	136.700000	521.600000	768	19.040000	0.055400	0.017500	
BGE-m3	567.800000	2165.800000	1024	47.990000	0.069000	0.028500	

## 6. Visualización comparativa

```
In [10]: fig, axes = plt.subplots(1, 3, figsize=(18, 6))

models_names = list(eval_results.keys())
colors = plt.cm.Set2(np.linspace(0, 1, len(models_names)))

# Gráfico 1: Métricas de retrieval
metrics = ['mean_precision', 'mean_recall', 'mrr']
labels = ['Precision@5', 'Recall@5', 'MRR']
x = np.arange(len(labels))
width = 0.15
for i, name in enumerate(models_names):
    ev = eval_results[name]
    vals = [ev[m] for m in metrics]
    axes[0].bar(x + i * width, vals, width, label=name, color=colors[i])
axes[0].set_xticks(x + width * (len(models_names) - 1) / 2)
axes[0].set_xticklabels(labels)
axes[0].set_ylim(0, 1.05)
axes[0].set_title('Calidad de Retrieval (español)')
axes[0].legend(fontsize=8)

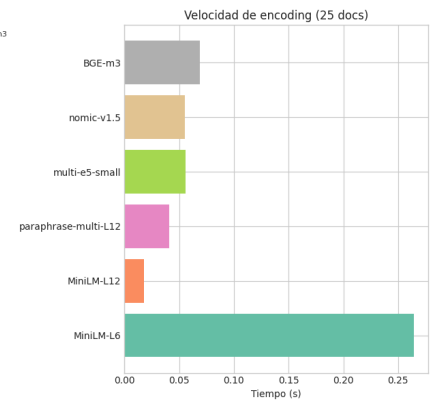
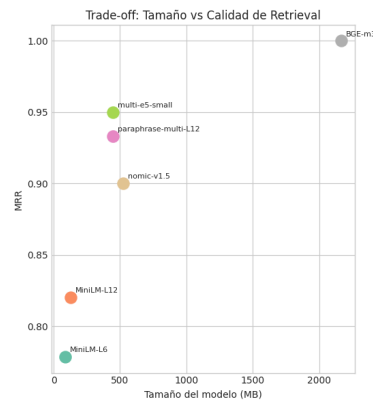
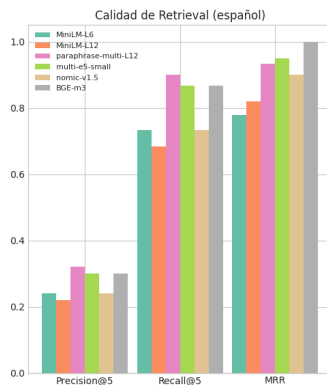
# Gráfico 2: Tamaño vs MRR (scatter)
for i, name in enumerate(models_names):
    r = results[name]
    ev = eval_results[name]
    axes[1].scatter(r['model_mb'], ev['mrr'], s=150, color=colors[i], label=
    axes[1].annotate(name, (r['model_mb'], ev['mrr']), textcoords='offset po
    xytext=(5, 5), fontsize=8)
axes[1].set_xlabel('Tamaño del modelo (MB)')
axes[1].set_ylabel('MRR')
axes[1].set_title('Trade-off: Tamaño vs Calidad de Retrieval')

# Gráfico 3: Tiempo de encoding del corpus
encode_times = [results[n]['corpus_encode_s'] for n in models_names]
axes[2].barh(models_names, encode_times, color=colors)
```



```
axes[2].set_xlabel('Tiempo (s)')
axes[2].set_title('Velocidad de encoding (25 docs)')

plt.tight_layout()
plt.show()
```



## 7. Detalle por query

Veamos qué documentos recupera cada modelo para cada query, para entender dónde fallan.

```
In [7]: # Detalle: para cada query, qué recuperó cada modelo
for qi, (query_text, relevant) in enumerate(queries):
    print(f'\nQ{qi}: "{query_text}"')
    print(f' Relevantes esperados: {relevant}')
    for name in results:
        ev = eval_results[name]
        pq = ev['per_query'][qi]
        retrieved = pq['retrieved']
        hits = [f'✓{idx}' if idx in set(relevant) else f'{idx}' for idx in
        print(f' {name:25s} → [{" ".join(hits)}] P={pq["precision@k"]:.2f}')
```

Q0: "¿Cómo manejan los lenguajes de programación la memoria?"

Relevantes esperados: [0, 2]

MiniLM-L6	→ [✓0 3 8 16 ✓2]	P=0.40 R=1.00 RR=1.00
MiniLM-L12	→ [✓0 3 15 ✓2 14]	P=0.40 R=1.00 RR=1.00
paraphrase-multi-L12	→ [✓0 ✓2 14 4 7]	P=0.40 R=1.00 RR=1.00
multi-e5-small	→ [✓0 3 ✓2 4 7]	P=0.40 R=1.00 RR=1.00
nomic-v1.5	→ [✓0 ✓2 3 8 18]	P=0.40 R=1.00 RR=1.00
BGE-m3	→ [✓0 ✓2 18 1 3]	P=0.40 R=1.00 RR=1.00

Q1: "¿Qué es el control de versiones para código fuente?"

Relevantes esperados: [4]

MiniLM-L6	→ [✓4 16 15 6 11]	P=0.20 R=1.00 RR=1.00
MiniLM-L12	→ [✓4 15 14 18 11]	P=0.20 R=1.00 RR=1.00
paraphrase-multi-L12	→ [✓4 14 0 2 23]	P=0.20 R=1.00 RR=1.00
multi-e5-small	→ [✓4 0 1 3 2]	P=0.20 R=1.00 RR=1.00
nomic-v1.5	→ [✓4 16 9 12 8]	P=0.20 R=1.00 RR=1.00
BGE-m3	→ [✓4 1 3 0 2]	P=0.20 R=1.00 RR=1.00

Q2: "¿Cómo producen energía las plantas a partir de la luz?"

Relevantes esperados: [5]

MiniLM-L6	→ [✓5 20 14 8 15]	P=0.20 R=1.00 RR=1.00
MiniLM-L12	→ [✓5 20 15 8 14]	P=0.20 R=1.00 RR=1.00
paraphrase-multi-L12	→ [✓5 14 15 9 20]	P=0.20 R=1.00 RR=1.00
multi-e5-small	→ [✓5 8 20 13 10]	P=0.20 R=1.00 RR=1.00
nomic-v1.5	→ [✓5 20 8 10 21]	P=0.20 R=1.00 RR=1.00
BGE-m3	→ [✓5 9 10 13 12]	P=0.20 R=1.00 RR=1.00

Q3: "Cuéntame sobre física de partículas y mecánica cuántica"

Relevantes esperados: [7, 8]

MiniLM-L6	→ [✓8 14 20 21 13]	P=0.20 R=0.50 RR=1.00
MiniLM-L12	→ [✓8 13 14 12 9]	P=0.20 R=0.50 RR=1.00
paraphrase-multi-L12	→ [✓8 6 12 18 14]	P=0.20 R=0.50 RR=1.00
multi-e5-small	→ [✓8 6 21 5 12]	P=0.20 R=0.50 RR=1.00
nomic-v1.5	→ [✓8 21 6 14 5]	P=0.20 R=0.50 RR=1.00
BGE-m3	→ [✓8 6 0 23 13]	P=0.20 R=0.50 RR=1.00

Q4: "¿Cómo funciona la fermentación del pan?"

Relevantes esperados: [10, 14]

MiniLM-L6	→ [✓10 ✓14 12 20 13]	P=0.40 R=1.00 RR=1.00
MiniLM-L12	→ [✓14 ✓10 12 11 15]	P=0.40 R=1.00 RR=1.00
paraphrase-multi-L12	→ [✓10 ✓14 12 13 11]	P=0.40 R=1.00 RR=1.00
multi-e5-small	→ [✓10 ✓14 12 13 7]	P=0.40 R=1.00 RR=1.00
nomic-v1.5	→ [✓10 ✓14 12 13 19]	P=0.40 R=1.00 RR=1.00
BGE-m3	→ [✓10 ✓14 13 9 5]	P=0.40 R=1.00 RR=1.00

Q5: "¿Qué hace que la comida sepa mejor al cocinarla a fuego alto?"

Relevantes esperados: [12, 13]

MiniLM-L6	→ [ 15 14 ✓12 11 10]	P=0.20 R=0.50 RR=0.33
MiniLM-L12	→ [ 15 10 14 11 8]	P=0.00 R=0.00 RR=0.00
paraphrase-multi-L12	→ [✓12 14 10 11 ✓13]	P=0.40 R=1.00 RR=1.00
multi-e5-small	→ [✓12 14 ✓13 10 11]	P=0.40 R=1.00 RR=1.00
nomic-v1.5	→ [✓12 14 8 10 15]	P=0.20 R=0.50 RR=1.00
BGE-m3	→ [✓12 14 11 ✓13 0]	P=0.40 R=1.00 RR=1.00

Q6: "¿Cómo puedo hacer crecer mis ahorros con el tiempo?"

Relevantes esperados: [15, 16, 19]

MiniLM-L6	→ [✓15 6 11 14 20]	P=0.20 R=0.33 RR=1.00
MiniLM-L12	→ [✓15 11 14 6 8]	P=0.20 R=0.33 RR=1.00
paraphrase-multi-L12	→ [✓15 17 ✓16 ✓19 14]	P=0.60 R=1.00 RR=1.00
multi-e5-small	→ [✓15 18 ✓16 17 6]	P=0.40 R=0.67 RR=1.00
nomic-v1.5	→ [✓15 14 6 12 24]	P=0.20 R=0.33 RR=1.00
BGE-m3	→ [✓15 ✓16 18 17 14]	P=0.40 R=0.67 RR=1.00

Q7: "¿Qué afecta la economía y los precios?"

Relevantes esperados: [17, 18]

MiniLM-L6	→ [ 20 15 14 ✓18 6]	P=0.20 R=0.50 RR=0.25
MiniLM-L12	→ [✓18 15 14 20 24]	P=0.20 R=0.50 RR=1.00
paraphrase-multi-L12	→ [✓17 19 16 15 23]	P=0.20 R=0.50 RR=1.00
multi-e5-small	→ [✓17 23 16 9 ✓18]	P=0.40 R=1.00 RR=1.00
nomic-v1.5	→ [✓17 14 ✓18 12 19]	P=0.40 R=1.00 RR=1.00
BGE-m3	→ [✓17 ✓18 23 21 15]	P=0.40 R=1.00 RR=1.00

Q8: "Háblame de las partes más profundas del océano"

Relevantes esperados: [24]

MiniLM-L6	→ [✓24 6 8 23 15]	P=0.20 R=1.00 RR=1.00
MiniLM-L12	→ [✓24 23 8 15 20]	P=0.20 R=1.00 RR=1.00
paraphrase-multi-L12	→ [✓24 23 22 20 13]	P=0.20 R=1.00 RR=1.00
multi-e5-small	→ [✓24 23 20 21 22]	P=0.20 R=1.00 RR=1.00
nomic-v1.5	→ [✓24 23 13 8 21]	P=0.20 R=1.00 RR=1.00
BGE-m3	→ [✓24 23 21 0 9]	P=0.20 R=1.00 RR=1.00

Q9: "¿Cómo cambia la superficie de la Tierra con el tiempo?"

Relevantes esperados: [21, 22]

MiniLM-L6	→ [ 6 15 14 20 ✓21]	P=0.20 R=0.50 RR=0.20
MiniLM-L12	→ [ 15 6 24 14 ✓22]	P=0.20 R=0.50 RR=0.20
paraphrase-multi-L12	→ [ 23 6 ✓21 ✓22 17]	P=0.40 R=1.00 RR=0.33
multi-e5-small	→ [ 6 ✓21 23 8 5]	P=0.20 R=0.50 RR=0.50
nomic-v1.5	→ [ 6 14 8 12 24]	P=0.00 R=0.00 RR=0.00
BGE-m3	→ [✓21 9 23 6 5]	P=0.20 R=0.50 RR=1.00

## 8. Visualización t-SNE: ¿Cómo agrupa cada modelo los documentos?

Proyectamos los embeddings del corpus a 2D con t-SNE. Cada color es una categoría. Un buen modelo debería agrupar documentos del mismo tema juntos.

```
In [8]: from sklearn.manifold import TSNE

categories = ['Programación'] * 5 + ['Ciencia'] * 5 + ['Cocina'] * 5 + ['Ficción'] * 5
cat_colors = {'Programación': '#e74c3c', 'Ciencia': '#3498db', 'Cocina': '#2ecc71', 'Ficción': '#9b59b6'}

n_models = len(results)
fig, axes = plt.subplots(2, 3, figsize=(18, 12))
axes = axes.flatten()

for i, (name, res) in enumerate(results.items()):
    embs = res['corpus_embs']
    tsne = TSNE(n_components=2, random_state=42, perplexity=8)
    coords = tsne.fit_transform(embs)
```

```

ax = axes[i]
for cat in cat_colors:
    mask = [c == cat for c in categories]
    ax.scatter(coords[np.array(mask), 0], coords[np.array(mask), 1],
               c=cat_colors[cat], label=cat, s=80, alpha=0.8, edgecolors='k')
ax.set_title(f'{name} ({res["dim"]}d, {res["n_params_m"]}M)', fontsize=12)
ax.set_xticks([]); ax.set_yticks([])
if i == 0:
    ax.legend(fontsize=8, loc='upper left')

# Ocultar ejes sobrantes
for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)

plt.suptitle('t-SNE: Agrupación de documentos por categoría (corpus español)')
plt.tight_layout()
plt.show()

```

t-SNE: Agrupación de documentos por categoría (corpus español)



## 9. Conclusiones

Ejecuta todas las celdas arriba y compara:

- **Modelos solo-inglés** (MiniLM-L6, L12): muy rápidos y livianos, pero ¿entienden español?
- **Multilingüe ligero** (paraphrase-multi-L12, multi-e5-small): buen balance
- **Nomic v1.5**: más grande pero con Matryoshka — ¿compensa el tamaño extra?
- **BGE-m3**: el más pesado — ¿la diferencia en retrieval justifica 4x el tamaño?

La tabla resumen y los gráficos dan la respuesta concreta para **tu** caso de uso.