

In []:

Guardar multilingual-e5-small en local y benchmark de carga

Guardamos el modelo en `.local/models/` (gitignored) y comparamos:

- Carga desde HuggingFace cache vs carga desde path local
- Tiempo de primer encoding tras carga
- Verificación de que los embeddings son idénticos

In [1]:

```
import os, time, gc
import numpy as np
from sentence_transformers import SentenceTransformer
import psutil

FN_ROOT = os.environ.get('FN_REGISTRY_ROOT', '/home/lucas/fn_registry')
MODEL_ID = 'intfloat/multilingual-e5-small'
LOCAL_PATH = os.path.join(FN_ROOT, '.local', 'models', 'multilingual-e5-small')

def ram_mb():
    return psutil.Process(os.getpid()).memory_info().rss / (1024 * 1024)

print(f'FN_REGISTRY_ROOT: {FN_ROOT}')
print(f'Destino local: {LOCAL_PATH}')
print(f'RAM actual: {ram_mb():.0f} MB')
```

```
FN_REGISTRY_ROOT: /home/lucas/fn_registry
Destino local: /home/lucas/fn_registry/.local/models/multilingual-e5-small
RAM actual: 833 MB
```

1. Guardar modelo en .local/models/

In [2]:

```
# Cargar desde HF cache y guardar localmente
print('Cargando modelo desde HuggingFace cache...')
t0 = time.perf_counter()
model = SentenceTransformer(MODEL_ID)
hf_load_time = time.perf_counter() - t0
print(f' Cargado en {hf_load_time:.2f}s')

# Guardar en .local/models/
print(f'Guardando en {LOCAL_PATH}...')
t0 = time.perf_counter()
model.save(LOCAL_PATH)
save_time = time.perf_counter() - t0
print(f' Guardado en {save_time:.2f}s')

# Verificar tamaño en disco
```

```

total_size = 0
for dirpath, dirnames, filenames in os.walk(LOCAL_PATH):
    for f in filenames:
        fp = os.path.join(dirpath, f)
        total_size += os.path.getsize(fp)
print(f' Tamaño en disco: {total_size / (1024*1024):.1f} MB')

# Listar archivos guardados
for dirpath, dirnames, filenames in os.walk(LOCAL_PATH):
    level = dirpath.replace(LOCAL_PATH, '').count(os.sep)
    indent = ' ' * level
    subdir = os.path.basename(dirpath)
    print(f' {indent}{subdir}/')
    for f in sorted(filenames):
        size = os.path.getsize(os.path.join(dirpath, f))
        print(f' {indent} {f} ({size/(1024*1024):.1f} MB)')

del model
gc.collect()

```

Cargando modelo desde HuggingFace cache...

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%| | 0/199 [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.

Cargado en 6.43s

Guardando en /home/lucas/fn_registry/.local/models/multilingual-e5-small...

Writing model shards: 0%| | 0/1 [00:00<?, ?it/s]

Guardado en 2.33s

Tamaño en disco: 465.6 MB

multilingual-e5-small/

README.md (0.5 MB)

config.json (0.0 MB)

config_sentence_transformers.json (0.0 MB)

model.safetensors (448.8 MB)

modules.json (0.0 MB)

sentence_bert_config.json (0.0 MB)

tokenizer.json (16.3 MB)

tokenizer_config.json (0.0 MB)

1_Pooling/

config.json (0.0 MB)

2_Normalize/

Out[2]: 4063

2. Benchmark: carga local vs HF cache

Cargamos 5 veces de cada fuente para tener estadísticas estables.

In [3]: N_RUNS = 5

```
def bench_load(source, path, runs=N_RUNS):
    """Carga modelo N veces, retorna tiempos y RAM."""
    times = []
    rams = []
    for i in range(runs):
        gc.collect()
        ram_before = ram_mb()
        t0 = time.perf_counter()
        m = SentenceTransformer(path)
        elapsed = time.perf_counter() - t0
        ram_after = ram_mb()
        times.append(elapsed)
        rams.append(ram_after - ram_before)
    del m
    gc.collect()
    return {
        'source': source,
        'times': times,
        'mean_time': np.mean(times),
        'std_time': np.std(times),
        'min_time': np.min(times),
        'mean_ram_delta': np.mean(rams),
    }

print(f'Benchmark de carga ({N_RUNS} iteraciones cada uno)...')
print()

# Carga desde HF cache
print('→ Cargando desde HuggingFace cache...')
hf_bench = bench_load('HF cache', MODEL_ID)
print(f'  Media: {hf_bench["mean_time"]:.3f}s ± {hf_bench["std_time"]:.3f}s

# Carga desde path local
print('→ Cargando desde .local/models/...')
local_bench = bench_load('Local path', LOCAL_PATH)
print(f'  Media: {local_bench["mean_time"]:.3f}s ± {local_bench["std_time"]:.3f}s

speedup = hf_bench['mean_time'] / local_bench['mean_time']
print(f'\nSpeedup local vs HF cache: {speedup:.2f}x')
```

Benchmark de carga (5 iteraciones cada uno)...

→ Cargando desde HuggingFace cache...

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.

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.

```
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.

```
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.

```
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.

```
Media: 4.134s ± 0.211s (min: 3.797s)
```

```
→ Cargando desde .local/models/...
```

```
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]
```

```
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]
```

```
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]
```

```
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]
```

```
Loading weights: 0%| | 0/199 [00:00<?, ?it/s]
```

```
Media: 1.821s ± 0.138s (min: 1.563s)
```

```
Speedup local vs HF cache: 2.27x
```

3. Verificación: embeddings idénticos

Confirmamos que el modelo local produce exactamente los mismos embeddings que el de HF cache.

```
In [4]: test_texts = [  
        'passage: La inteligencia artificial transforma la industria tecnológica'
```

```

    'passage: Los mercados financieros reaccionan a las decisiones de políti
    'passage: El aprendizaje profundo requiere grandes cantidades de datos e
    'query: ¿Cómo funciona el machine learning?',
    'query: ¿Qué afecta los precios de las acciones?',
]

# Cargar ambos modelos
model_hf = SentenceTransformer(MODEL_ID)
model_local = SentenceTransformer(LOCAL_PATH)

# Generar embeddings
emb_hf = model_hf.encode(test_texts, normalize_embeddings=True)
emb_local = model_local.encode(test_texts, normalize_embeddings=True)

# Comparar
max_diff = np.max(np.abs(emb_hf - emb_local))
mean_diff = np.mean(np.abs(emb_hf - emb_local))
are_equal = np.allclose(emb_hf, emb_local, atol=1e-6)

print(f'Max diferencia absoluta: {max_diff:.2e}')
print(f'Mean diferencia absoluta: {mean_diff:.2e}')
print(f'¿Son idénticos (atol=1e-6)? {"✓ SÍ" if are_equal else "x NO"}')

# Cosine similarity entre pares correspondientes
for i, text in enumerate(test_texts):
    sim = np.dot(emb_hf[i], emb_local[i])
    print(f' [{i}] cosine={sim:.10f} | {text[:60]}')

del model_hf, model_local
gc.collect()

```

```
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.

```
Loading weights: 0%|          | 0/199 [00:00<?, ?it/s]
```

```
Max diferencia absoluta: 0.00e+00
```

```
Mean diferencia absoluta: 0.00e+00
```

```
¿Son idénticos (atol=1e-6)? ✓ SÍ
```

```
[0] cosine=1.0000000000 | passage: La inteligencia artificial transforma l
a industria
```

```
[1] cosine=1.0000000000 | passage: Los mercados financieros reaccionan a l
as decisiones
```

```
[2] cosine=1.0000000000 | passage: El aprendizaje profundo requiere grande
s cantidades
```

```
[3] cosine=1.0000000000 | query: ¿Cómo funciona el machine learning?
```

```
[4] cosine=1.0000000000 | query: ¿Qué afecta los precios de las acciones?
```

```
Out[4]: 6961
```

4. Visualización de tiempos de carga

```
In [5]: import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Box plot de tiempos
data = [hf_bench['times'], local_bench['times']]
labels = ['HF cache', 'Local (.local/)']
colors = ['#e74c3c', '#2ecc71']

bp = axes[0].boxplot(data, labels=labels, patch_artist=True)
for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)
axes[0].set_ylabel('Tiempo de carga (s)')
axes[0].set_title('Distribución de tiempos de carga')

# Bar chart comparativo
means = [hf_bench['mean_time'], local_bench['mean_time']]
stds = [hf_bench['std_time'], local_bench['std_time']]
bars = axes[1].bar(labels, means, yerr=stds, color=colors, alpha=0.8, capsiz=5)
for bar, mean in zip(bars, means):
    axes[1].text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,
                  f'{mean:.3f}s', ha='center', fontsize=12, fontweight='bold')
axes[1].set_ylabel('Tiempo medio (s)')
axes[1].set_title(f'Carga media ({N_RUNS} runs) — speedup: {speedup:.2f}x')

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

/tmp/ipykernel_18429/1763775165.py:10: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.

```
bp = axes[0].boxplot(data, labels=labels, patch_artist=True)
```

