```
!pip install ijson
import ijson
import torch
from sentence_transformers import SentenceTransformer
import gc

# ============================================================================
# Configuration
# ============================================================================
INPUT_FILE = '/content/drive/MyDrive/BM25 Files/hover_dev_bm25_top100.json'
MODEL_NAME = 'sentence-transformers/multi-qa-mpnet-base-dot-v1'
BATCH_SIZE = 16
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'

# ============================================================================
# Initialize Model First
# ============================================================================
print(f"Loading model: {MODEL_NAME}")
model = SentenceTransformer(MODEL_NAME, device=DEVICE)
print(f"Using device: {DEVICE}")

# ============================================================================
# Count claims using streaming (no full load)
# ============================================================================
print("\nCounting claims...")
claim_count = 0
with open(INPUT_FILE, 'rb') as f:
    parser = ijson.kvitems(f, '')
    for _ in parser:
        claim_count += 1

print(f"Total claims: {claim_count}")
print("Memory safe - never loaded full file")
```

```
Collecting ijson
  Downloading ijson-3.4.0.post0-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (23 kB)
Downloading ijson-3.4.0.post0-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (149 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.0/149.0 kB 6.4 MB/s eta 0:00:00
Installing collected packages: ijson
Successfully installed ijson-3.4.0.post0
Loading model: sentence-transformers/multi-qa-mpnet-base-dot-v1
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Using device: cuda

Counting claims...
Total claims: 4000
Memory safe - never loaded full file
```

```
# Debug: check structure
with open(INPUT_FILE, 'rb') as f:
    parser = ijson.kvitems(f, '')
    claim_id, claim_data = next(parser)
    print("Claim keys:", claim_data.keys())
```

```
    print("First doc keys:", claim_data['retrieved_docs'][0].keys())
    print("First doc sample:", claim_data['retrieved_docs'][0])
```

```
ensus Bureau and currently Deputy Undersecretary for Economic Affairs at the US Department of Commerce.'], 'score': Decimal('115.60873'), 'url': 'https://en.wikipedia.org/wiki?curid=1459568'}
```

```
from tqdm import tqdm
import json
from decimal import Decimal

# ============================================================================
# Re-ranking Function
# ============================================================================
def rerank_documents(claim_text, documents, model, batch_size=16):
    """Re-rank documents using dense retrieval."""
    # Encode claim
    claim_embedding = model.encode(claim_text, convert_to_tensor=True, show_progress_bar=False)

    # Extract document texts (join sentences)
    doc_texts = [' '.join(doc['sentences']) for doc in documents]

    # Batch encode documents
    doc_embeddings = model.encode(
        doc_texts,
        batch_size=batch_size,
        convert_to_tensor=True,
        show_progress_bar=False
    )

    # Compute similarity scores (dot product)
    scores = torch.matmul(doc_embeddings, claim_embedding)
    scores = scores.cpu().numpy()

    # Add dense scores to documents
    for i, doc in enumerate(documents):
        doc['dense_score'] = float(scores[i])
        doc['bm25_score'] = float(doc['score'])  # Convert Decimal to float

    # Sort by dense score (descending)
    reranked_docs = sorted(documents, key=lambda x: x['dense_score'], reverse=True)

    # Clean up
    del claim_embedding, doc_embeddings, scores
    torch.cuda.empty_cache()

    return reranked_docs

# ============================================================================
# Custom JSON encoder for Decimal
# ============================================================================
class DecimalEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Decimal):
            return float(obj)
        return super(DecimalEncoder, self).default(obj)

# ============================================================================
```

```
# Process with Streaming (Resume from checkpoint)
# ============================================================================
OUTPUT_FILE = '/content/drive/MyDrive/BM25 Files/hover_dev_dense_reranked_top100.json'
SAVE_EVERY = 50

print("\nRe-ranking documents...")

# Load checkpoint if exists
try:
    with open(OUTPUT_FILE + '.tmp', 'r') as f:
        reranked_results = json.load(f)
    print(f"Resuming: Loaded {len(reranked_results)} already processed claims")
except:
    reranked_results = {}
    print("Starting fresh")

processed = len(reranked_results)

with open(INPUT_FILE, 'rb') as f:
    parser = ijson.kvitems(f, '')

    for claim_id, claim_data in tqdm(parser, total=claim_count, desc="Processing"):
        # Skip already processed
        if claim_id in reranked_results:
            continue

        claim_text = claim_data['claim']
        retrieved_docs = claim_data['retrieved_docs']

        # Re-rank
        reranked_docs = rerank_documents(claim_text, retrieved_docs, model, BATCH_SIZE)

        # Store
        reranked_results[claim_id] = {
            'claim': claim_text,
            'retrieved_docs': reranked_docs
        }

        processed += 1

        # Periodic save
        if processed % SAVE_EVERY == 0:
            with open(OUTPUT_FILE + '.tmp', 'w') as out:
                json.dump(reranked_results, out, cls=DecimalEncoder)
            gc.collect()

# Final save
print(f"\nSaving to {OUTPUT_FILE}")
with open(OUTPUT_FILE, 'w') as f:
    json.dump(reranked_results, f, indent=2, cls=DecimalEncoder)

print("✓ Complete!")
```

```
Re-ranking documents...
Starting fresh
Processing: 100%|██████████| 4000/4000 [3:41:23<00:00,  3.32s/it]

Saving to /content/drive/MyDrive/BM25 Files/hover_dev_dense_reranked_top100.json
```

```
✓ Complete!
```

```
# ============================================================================
# Compute Metrics
# ============================================================================
print("\n" + "="*60)
print("METRIC EVALUATION")
print("="*60)

# Build ground truth from BM25 file (it has supporting_facts)
print("Building ground truth from BM25 results...")
gt_map = {}

with open(INPUT_FILE, 'rb') as f:
    parser = ijson.kvitems(f, '')
    for claim_id, claim_data in parser:
        supporting_facts = set()
        for fact in claim_data.get('supporting_facts', []):
            # Format: [title, sentence_id]
            supporting_facts.add(f"{fact[0]}_{fact[1]}")
        gt_map[claim_id] = supporting_facts

print(f"Loaded ground truth for {len(gt_map)} claims")

# Compute metrics
def compute_metrics(results, ground_truth, k=10):
    total_retrieved = 0
    total_relevant = 0
    claims_with_relevant = 0

    for claim_id, data in results.items():
        if claim_id not in ground_truth:
            continue

        relevant_docs = ground_truth[claim_id]
        if not relevant_docs:
            continue

        # Build set using title_sentenceID format
        top_k_docs = set()
        for doc in data['retrieved_docs'][:k]:
            title = doc['title']
            # Add all sentences from this doc
            for sent_id in range(len(doc['sentences'])):
                top_k_docs.add(f"{title}_{sent_id}")

        retrieved_relevant = top_k_docs & relevant_docs

        total_retrieved += len(retrieved_relevant)
        total_relevant += len(relevant_docs)

        if len(retrieved_relevant) > 0:
            claims_with_relevant += 1

    recall = total_retrieved / total_relevant if total_relevant > 0 else 0
    coverage = claims_with_relevant / len(results) if len(results) > 0 else 0

    return recall, coverage
```

```
# Re-run metrics
print("\nBM25 (Original) Metrics:")
bm25_recall, bm25_coverage = compute_metrics(bm25_results, gt_map, k=10)
print(f"  Recall@10:   {bm25_recall*100:.2f}%")
print(f"  Coverage@10: {bm25_coverage*100:.2f}%")

print("\nDense Re-ranking Metrics:")
dense_recall, dense_coverage = compute_metrics(reranked_results, gt_map, k=10)
print(f"  Recall@10:   {dense_recall*100:.2f}%")
print(f"  Coverage@10: {dense_coverage*100:.2f}%")

print("\nImprovement:")
print(f"  Recall@10:   {(dense_recall - bm25_recall)*100:+.2f}pp")
print(f"  Coverage@10: {(dense_coverage - bm25_coverage)*100:+.2f}pp")
```

```
============================================================
METRIC EVALUATION
============================================================
Building ground truth from BM25 results...
Loaded ground truth for 4000 claims

BM25 (Original) Metrics:
  Recall@10:   33.76%
  Coverage@10: 70.65%

Dense Re-ranking Metrics:
  Recall@10:   43.24%
  Coverage@10: 83.97%

Improvement:
  Recall@10:   +9.48pp
  Coverage@10: +13.32pp
```