SOFTWARE, TECH

HOW TO RUN LLAMA ON YOUR LOCAL PDFS

14.06.2025

I needed this urgently for indexing PDFs as Spotlight on the Mac is highly erratic after all this years.

Anything LLM seemed the most promising approach with an easy to use GUI and being well documented. But indexing failed after several hours, so I went on with LM Studio. Also this installation turned out to be more complicated than expected due to library “dependency hell” and version mismatch spiralling…

1. Download and install LM Studio
2. From inside LM Studio download your preferred model
3. Index your PDFs in batches of 1,000 using the Python script below
4. Combine indices and run queries against the full index

30.000 PDFs result in a 4G index while the system is unfortunately not very responsive (yet)

index.py

```
import os
from tqdm import tqdm
from multiprocessing import get_context, TimeoutError
from pdfminer.high_level import extract_text
from sentence_transformers import SentenceTransformer
from llama_index.core import VectorStoreIndex, Document, Settings
from llama_index.llms.ollama import Ollama
from llama_index.embeddings.huggingface import HuggingFaceEmbedding

# === CONFIG ===
os.environ["TOKENIZERS_PARALLELISM"] = "false"
PDF_DIR = "/Users/xxx/Documents"
```

```
INDEX_BASE_DIR = "/Users/xxx/Store"
MODEL_NAME = "sentence-transformers/all-MiniLM-L6-v2"
LLM_API_BASE = "http://localhost:11434/v1"
CHUNK_SIZE = 10000
START_CHUNK = 1
END_CHUNK = 1
BAD_LOG = os.path.join(INDEX_BASE_DIR, "bad_files.txt")
TIMEOUT_SECONDS = 60

# === Step 1: Get all PDFs ===
def get_all_pdfs(path):
    pdf_files = []
    for root, _, files in os.walk(path):
        for file in files:
            if file.lower().endswith(".pdf"):
                pdf_files.append(os.path.join(root, file))
    return sorted(pdf_files)

# === Step 2: Timeout-safe PDF parsing ===
def parse_pdf_safe(path):
    try:
        text = extract_text(path)
        return {"success": True, "doc": Document(text=text,
metadata={"file_path": path})}
    except Exception as e:
        return {"success": False, "file_path": path, "error": str(e)}

def parse_pdfs_parallel(paths, timeout=TIMEOUT_SECONDS):
    documents = []
    print(f"Parsing {len(paths)} PDFs (timeout: {timeout}s each)...")

    ctx = get_context("spawn")
    pool = ctx.Pool(processes=os.cpu_count())

    try:
        for path in tqdm(paths, desc="Parsing PDFs"):
            async_result = pool.apply_async(parse_pdf_safe, (path,))
            try:
                result = async_result.get(timeout=timeout)
                if result["success"]:
                    documents.append(result["doc"])
```

```
                else:
                    print(f"{result['file_path']}: {result['error']}")
                    with open(BAD_LOG, "a") as log:
                        log.write(f"FAIL: {result['file_path']} ::
{result['error']}\n")
            except TimeoutError:
                print(f"Timeout: {path}")
                with open(BAD_LOG, "a") as log:
                    log.write(f"TIMEOUT: {path}\n")
    finally:
        pool.terminate()
        pool.join()

    return documents

# === Step 3: Build and save index ===
def build_index(documents, index_dir):
    print(f"Indexing {len(documents)} documents → {index_dir}")
    embed_model =
HuggingFaceEmbedding(model_name="sentence-transformers/all-MiniLM-L6-v
2")
    # llm = OpenAI(api_base="http://localhost:11434/v1",
api_key="lm-studio")
    llm = Ollama(model="llama3")

    Settings.embed_model = embed_model
    Settings.llm = llm

    index = VectorStoreIndex.from_documents(documents)
    index.storage_context.persist(persist_dir=index_dir)
    print(f"Index saved to {index_dir}")

# === MAIN ===
if __name__ == "__main__":
    all_pdfs = get_all_pdfs(PDF_DIR)
    total = len(all_pdfs)
    total_chunks = (total + CHUNK_SIZE - 1) // CHUNK_SIZE

    os.makedirs(INDEX_BASE_DIR, exist_ok=True)
    open(BAD_LOG, "w").close()  # clear previous log
```

```
    for chunk_num in range(START_CHUNK, END_CHUNK + 1):
        start = (chunk_num - 1) * CHUNK_SIZE
        end = min(start + CHUNK_SIZE, total)
        chunk_paths = all_pdfs[start:end]

        index_dir = os.path.join(INDEX_BASE_DIR, f"part_{chunk_num}")
        if os.path.exists(index_dir):
            print(f"hunk {chunk_num} already exists, skipping:
{index_dir}")
            continue

        print(f"\nProcessing chunk {chunk_num}/{total_chunks} → files
{start+1} to {end}")
        docs = parse_pdfs_parallel(chunk_paths)
        build_index(docs, index_dir)
```

query.py

```
import os
from llama_index.core import StorageContext, load_index_from_storage,
VectorStoreIndex, Document, Settings
from llama_index.llms.openai import OpenAI
from llama_index.embeddings.huggingface import HuggingFaceEmbedding

# === CONFIG ===
INDEX_PARTS_DIR = "/Users/xxx/Store"
MERGED_INDEX_DIR = os.path.join(INDEX_PARTS_DIR, "merged")

PART_DIRS = [
    os.path.join(INDEX_PARTS_DIR, d)
    for d in os.listdir(INDEX_PARTS_DIR)
    if d.startswith("part_") and
os.path.isdir(os.path.join(INDEX_PARTS_DIR, d))
]

# === Setup LM Studio OpenAI-compatible API ===
os.environ["OPENAI_API_KEY"] = "not-needed"  # dummy value to bypass
API key checks
os.environ["OPENAI_API_BASE"] = "http://localhost:1234/v1"  # LM
Studio API base
```

```
# Embedding + LLM setup using LM Studio OpenAI API
Settings.embed_model =
HuggingFaceEmbedding(model_name="sentence-transformers/all-MiniLM-L6-v
2")
Settings.llm = OpenAI(
    model="llama-3.2-3b-instruct",   # or your LM Studio supported
model
    api_key="not-needed",    # dummy key, LM Studio ignores it
    api_base="http://localhost:1234/v1",
)

# === Load & collect all documents from partial indices ===
all_documents = []

for part_dir in sorted(PART_DIRS):
   print(f"Loading index from: {part_dir}")
   storage_context =
StorageContext.from_defaults(persist_dir=part_dir)
   index = load_index_from_storage(storage_context)

   for node in storage_context.docstore.docs.values():
       if isinstance(node, Document):
           all_documents.append(node)
       else:
           if hasattr(node, "get_content") and hasattr(node,
"metadata"):
               all_documents.append(Document(text=node.get_content(),
metadata=node.metadata))

print(f"Total documents to merge: {len(all_documents)}")

# === Build and persist merged index ===
merged_index = VectorStoreIndex.from_documents(all_documents)
merged_index.storage_context.persist(persist_dir=MERGED_INDEX_DIR)
print(f"Merged index saved to: {MERGED_INDEX_DIR}")

# === Load merged index for querying ===
storage = StorageContext.from_defaults(persist_dir=MERGED_INDEX_DIR)
index = load_index_from_storage(storage)
query_engine = index.as_query_engine()
```

```
# === Interactive query loop ===
print("\nAsk anything about the case. Type 'exit' or 'quit' to stop.")
while True:
    query = input("You: ").strip()
    if query.lower() in ("exit", "quit"):
        break
    try:
        response = query_engine.query(query)
        print(f"Response: {response}\n")
    except Exception as e:
        print(f"Error: {e}\n")
```

7 Aug 2025

So far, I used Deepseek inside LM studio as my model of choice. Since yesterday we can use also OpenAI's gpt-oss-20b which is basically OpenAI o4-mini.