IA Locale

Guide technique d'implémentation

De la théorie à la production

Stack technologique

Core ML

PyTorch 2.0+ - Framework principal

Transformers 4.35+ - Hugging Face

PEFT - LoRA/QLoRA

Accelerate - Multi-GPU

bitsandbytes - Quantization

Inference

llama.cpp - CPU/GPU GGUF

Ollama - Runtime simplifié

vLLM - Serving haute perf

TGI - Text Generation Inference

Vector DB & RAG

FAISS - Facebook AI Similarity

Qdrant - Production-ready

LangChain - Orchestration

LlamaIndex - Alternative

Architecture RAG détaillée

1. Document Processing

Chunking: RecursiveCharacterTextSplitter (512-1024 tokens, overlap 50-100)

2. Embedding Generation

Models: bge-base-en-v1.5, e5-base-v2, gte-base (384-768 dims)

3. Vector Indexing

FAISS: IndexFlatIP, IndexIVFFlat (nlist=100), IndexHNSW (M=32, efConstruction=200)

4. Retrieval Strategy

Top-k=5-10, MMR diversification, optional reranking (cross-encoder)

5. LLM Generation

Context injection + prompt engineering + citation extraction

Setup environnement

```
conda create -n ai-local python=3.11
pip install torch transformers accelerate
pip install sentence-transformers faiss-cpu
pip install langchain chromadb peft bitsandbytes
```

CUDA: pip install torch --index-url https://download.pytorch.org/whl/cu121

ROCm: pip install torch --index-url https://download.pytorch.org/whl/rocm5.7

Embeddings: Implémentation

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("BAAI/bge-base-en-v1.5")
embeddings = model.encode(texts, normalize_embeddings=True)
```

Modèles recommandés:

- bge-base-en-v1.5: 768d, MTEB 63.5
- e5-base-v2: 768d, multilingual
- gte-base: 768d, optimisé vitesse

FAISS: Indexation vectorielle

```
import faiss, numpy as np
d = 768  # dimension
index = faiss.IndexFlatIP(d)  # Inner Product
faiss.normalize_L2(embeddings)
index.add(embeddings)
D, I = index.search(query_emb, k=5)
```

Index types: Flat (exact), IVF (fast), HNSW (balanced)

Ollama: Inférence locale

```
# Installation
curl -fsSL https://ollama.com/install.sh | sh
# Pull model
ollama pull llama3.1:8b
# Run
ollama run llama3.1:8b
```

Modèles 8B: Llama3.1, Mistral, Phi-3 | **13B+:** Llama3.1, Mixtral

QLoRA: Fine-tuning efficace

```
from peft import LoraConfig, get_peft_model
config = LoraConfig(r=16, lora_alpha=32,
target_modules=["q_proj","v_proj"],
lora_dropout=0.05)
model = get_peft_model(base_model, config)
```

Params: r=8-64 (rank), alpha=2*r, lr=1e-4 to 2e-4, epochs=1-3

Quantization: GGUF

```
llama.cpp --model model.gguf -ngl 32 -c 4096
```

Formats: Q4_K_M (4.5GB), Q5_K_M (5.5GB), Q8_0 (8GB)

Trade-off: Q4 = -1% qualité, 4x moins VRAM

Prompt Engineering

System: Define role et contraintes

Context: Inject retrieved passages

Few-shot: 2-3 examples pour format

CoT: Chain-of-thought pour raisonnement

Evaluation Metrics

Retrieval: Precision@k, Recall@k, MRR

Generation: BLEU, ROUGE, BERTScore

E2E: Exact Match, F1, Human eval

Latency: p50, p95, p99 response time

Production Deployment

```
docker run -d -v ./models:/models \\
-p 11434:11434 ollama/ollama
```

Load balancing: nginx, traefik | **Scaling:** k8s, docker swarm

Monitoring & Logging

Metrics: Prometheus + Grafana

Logs: ELK stack, Loki

Tracing: OpenTelemetry, Jaeger

Alerts: AlertManager, PagerDuty

Troubleshooting

OOM: Reduce batch, use gradient checkpointing

Slow inference: Quantize, optimize context length

Poor quality: Better retrieval, reranking, prompt tuning

CUDA errors: Check drivers, PyTorch CUDA version

Glossaire technique

GGUF: GPT-Generated Unified Format

LoRA: Low-Rank Adaptation

QLoRA: Quantized LoRA

PEFT: Parameter-Efficient Fine-Tuning

vLLM: Efficient LLM Inference

FAISS: Facebook AI Similarity Search

HNSW: Hierarchical NSW graph

IVF: Inverted File Index

MMR: Maximal Marginal Relevance

MTEB: Massive Text Embed Benchmark