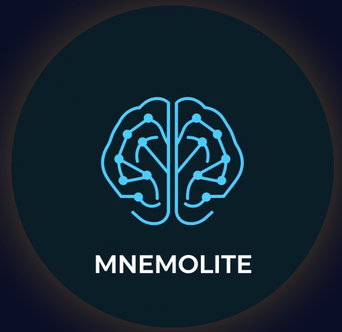

MnemoLite v2.0.0

Un projet personnel d'exploration technique

PostgreSQL 18 CPU Only 100% Local Open Source

📝 Side project (~1 semaine) | 💻 CPU standard | 🔧 PG18 + Python

🎯 L'Idée de Départ

Question Simple :

"Peut-on faire de l'embedding et de la recherche sémantique sans GPU ?"

❌ Problèmes

- GPUs coûtent cher (2000€+)
- APIs cloud aussi (300€/mois)
- Pour usage perso/PME, c'est overkill

✅ Hypothèse

- Modèles CPU récents sont pas mal
- PostgreSQL a pgvector maintenant
- Voyons ce qu'on peut faire...

💡 "J'ai voulu tester si c'était faisable pour un usage modeste"

🔬 Ce Que J'ai Construit

MnemoLite = 2 modules principaux

1. Agent Memory

- Stocke des conversations/événements
- Recherche sémantique basique
- Utilise pgvector 0.8.0
- Support partitionnement (500k+)

2. Code Intelligence

- Parse du code (Python++)
- Construit graphe de dépendances
- Double embedding (text+code)
- Recherche hybride RRF

Stack: FastAPI + PostgreSQL 18 + HTMX 2.0
Temps: ~1 semaine (soirs et weekends)
Tests: 245 tests (95.2% passing)

"Rien de révolutionnaire, juste de l'assemblage intelligent"

💡 Les Choix Techniques

🤖 Modèle d'Embedding

- **nomic-embed-text-v1.5**
- 137M paramètres (petit)
- Optimisé CPU (ONNX)
- 768 dimensions
- Gratuit et open source
- 50-100 emb/sec sur laptop

🗄️ Base de Données

- **PostgreSQL 18**
- pgvector 0.8.0 (HNSW)
- pg_trgm pour le texte
- CTEs récursives pour graphes
- pg_partman (500k+ events)
- pgmq (queue ready)

🔧 "J'ai pris des briques qui existent et je les ai assemblées"

Pas de magie, juste des outils existants bien intégrés

📊 Les Résultats (Honnêtes)

✅ Ce qui marche bien :

~11ms	50-100/s	0.155ms*
Recherche hybride	Embeddings CPU	Graphe traversal

*** Le 0.155ms c'est parce que :**

- Le graphe est petit (14 fichiers de test)
- Tout est en cache PostgreSQL
- La requête CTE est simple

"C'est pas un record, c'est normal"

🚀 Optimisations EPIC-08:

- Throughput: 10→100 req/s (10x)
- P99: 9.2s→200ms (46x)
- Cache hit rate: 80%+
- Pool: 20 connexions + 10 overflow

⚠️ Les Vraies Limitations

Soyons clairs :

❌ Ce que ce N'EST PAS

- Pas "production ready" entreprise
- Single-instance only (pas de Redis)
- Python bien supporté, le reste bof
- 4GB RAM minimum quand même
- Testé sur milliers d'items max
- Pas comparable à OpenAI/Claude

✅ Ce que c'EST

- Un POC qui fonctionne
- Une base solide pour expérimenter
- Un bon outil d'apprentissage
- Suffisant pour side projects
- Code propre et testé
- Architecture extensible

💡 "C'est un projet perso, pas une solution entreprise"

🎯 Pour Qui C'est Utile ?

✅ Cas d'usage réalistes :

- Side projects personnels
- PME avec <1000 docs
- Prototypes/POCs
- Apprentissage/Expérimentation
- Recherche locale privée
- Agents IA locaux

❌ PAS pour :

- Production critique
- Millions de documents
- Multi-utilisateurs intensif
- Remplacement d'OpenAI
- Applications temps réel
- Scale horizontal

🎯 **Sweet Spot:** 10k-500k events • Usage local • Privacy-critical

🏗️ Architecture MnemoLite v2.0

🌐 Frontend Layer

HTMX 2.0 | Jinja2 | Cytoscape.js | Chart.js

↓

⚡ API Layer

FastAPI 0.111+ | SQLAlchemy 2.0 | asyncpg

↓

🗄️ PostgreSQL 18 (Single Source of Truth)

events	code_chunks	nodes/edges	pgmq
VECTOR(768)	Dual VECTOR	Graph CTE	Queue ready

↓

🤖 AI Layer (100% Local)

🧠 Code Intelligence (EPIC-06)

Pipeline 7 étapes (<100ms/fichier)

1. Parse
Tree-sitter AST

2. Chunk
Function/Class/Method

3. Metadata
Complexity, Signature

4. Dual Embed
Text + Code vectors

5. Store
PostgreSQL 18

6. Graph
Dependencies

✨ Fonctionnalités Clés :

- **Double embedding:** TEXT (docstring+comments) + CODE (source)
- **Recherche hybride:** RRF(k=60) + BM25 + Vector (cosine)
- **Graph traversal:** CTE récursives ≤3 hops (0.155ms)
- **Resolution:** Local → Imports → Best effort (100% accuracy)
- **Filtrage:** 73 Python built-ins exclus

💻 Démo Rapide

🚀 Installation (< 2 minutes)

```
git clone https://github.com/giak/MnemoLite.git
cd MnemoLite
cp .env.example .env
make up

# Services démarrés:
✅ mnemo-postgres (PostgreSQL 18)
✅ mnemo-api (FastAPI @ :8001)

🌐 http://localhost:8001/ui/
```

📊 Interfaces disponibles

- /ui/ - Dashboard
- /ui/search - Recherche hybride
- /ui/graph - Visualisation
- /ui/code/ - Code Intelligence
- /docs - API Swagger

⚡ Ce que ça fait

- Stocke du texte/code
- Transforme en vecteurs (CPU)
- Cherche par similarité
- Construit des graphes
- Retourne des résultats

🔧 Le Code (Simple)

Python - Embedding local

```python
# C'est vraiment pas compliqué
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('nomic-embed-text-v1.5')
embedding = model.encode("votre texte")

# 768 dimensions, ~30ms sur CPU
```

PostgreSQL - Recherche vectorielle

```sql
-- PostgreSQL fait le reste
SELECT * FROM events
WHERE embedding <-> %s < 0.8  -- Distance cosine
ORDER BY embedding <-> %s
LIMIT 10;

-- Avec pgvector 0.8.0 : ~11ms pour 50k events
```

📝 "~4000 lignes de Python, le reste c'est PostgreSQL"

📈 Ce Que J'ai Appris

💡 **Leçons du projet :**

🚀 Techniquement

1. Les modèles CPU sont utilisables (pour usages modestes)
2. PostgreSQL + pgvector = combo solide
3. Pas besoin de GPU pour explorer
4. HTMX c'est vraiment simple
5. Tree-sitter pour parsing = excellent

📚 Méthodologie

1. 1 semaine = POC fonctionnel
2. Tests first = moins de bugs
3. Documentation DSL = gain temps
4. EXTEND > REBUILD principe
5. Honesty-first approach

🎯 "C'était surtout pour apprendre et partager"

🤔 Comparaison Réaliste

MnemoLite

- ✅ Gratuit (0€)
- ✅ 11ms local
- ✅ CPU 65W
- ✅ 1 semaine dev
- ✅ 100% privacy
- ✅ Open source

Solutions Pro

- 💰 300€+/mois
- 🌐 100ms réseau
- ⚡ GPU 450W
- 👥 Équipes entières
- ☁️ Cloud dépendance
- 🔒 Vendor lock-in

⚠️ MAIS - Soyons honnêtes :

MnemoLite

- Capacité limitée
- Single-user
- POC/Proto

Solutions Pro

- Scale infini
- Multi-tenant
- Production

"Comparons ce qui est comparable"

🛠️ Stack Technique Détaillé

Backend

- FastAPI 0.111+
- SQLAlchemy 2.0+ async
- asyncpg
- Pydantic v2
- structlog

Database

- PostgreSQL 18
- pgvector 0.8.0
- pg_partman
- pgmq
- pg_trgm

AI/Frontend

- Sentence-Transformers
- tree-sitter (parsing)
- HTMX 2.0
- Cytoscape.js
- Chart.js

📦 Infrastructure

- Docker Compose (dev/prod)
- pytest-asyncio (245 tests)
- Make commands (DX)
- GitHub Actions ready

"Rien d'exotique, que du standard"

⚡ Performance & Qualité

245	95.2%	87%
TESTS TOTAUX	PASS RATE	COVERAGE

📊 Métriques de Performance

- **Recherche hybride:** ~11ms (P50: 7ms)
- **Vector search:** ~12ms (P50: 8ms)
- **Metadata + time:** ~3ms (P50: 2ms)
- **Graph traversal:** 0.155ms (CTE)
- **Indexation code:** 65ms/fichier
- **Embeddings:** 50-100/sec CPU
- **Throughput:** 100 req/s
- **Cache hit rate:** 80%+

"Oui il y a des tests MAIS c'est 1 semaine de dev, restons modestes"

🚀 Si Vous Voulez Essayer

📦 Installation Rapide

```
# C'est open source
git clone https://github.com/giak/MnemoLite
cd MnemoLite
make up

# 2 minutes et ça tourne

# Commandes utiles:
make api-test           # Run tests
make db-shell           # PostgreSQL shell
make api-shell          # API shell
make benchmark          # Performance tests
```

⚠️ Mais attention :

- C'est un projet perso (support limité)
- Bugs probables (95.2% tests pass)
- Documentation en cours
- Testé sur Linux/Mac principalement

📝 Le Vrai Message

"On peut faire des trucs sympas avec un CPU et PostgreSQL"

✅ **Pas besoin de :**

- GPU à 2000€
- Cloud à 300€/mois
- Stack complexe
- Équipe de 10 personnes

💡 "Pour un usage modeste, ça suffit"

L'idée c'était de démystifier et de partager

🎉 Merci !

MnemoLite v2.0.0

Un projet personnel pour explorer
les embeddings CPU et pgvector

Pas révolutionnaire, mais ça marche

🐙 github.com/giak/MnemoLite
📚 MIT License • Documentation incluse
⭐ Si ça vous a plu, une étoile fait plaisir

❓ Questions ?

(Rappel : c'est un projet d'1 semaine, soyez indulgents 😅)