Objectif : Transformer ma mémoire vectorielle actuelle en un véritable système multi-couches, inspiré du fonctionnement biologique.

📅 Dernière version : Octobre 2025
📍 Auteur : [Yann Bucaille]

Phase 1

1. Mémoire à court terme (STM)

- Stockée directement en RAM / JSON temporaire.
- Contient les dernières **10-20 interactions**.
- S'efface ou se résume automatiquement à chaque "session terminée".

```python
class ShortTermMemory:
    def __init__(self):
        self.buffer = deque(maxlen=20)
    def push(self, entry): ...
    def summarize(self): ...
```

---

2. Mémoire à moyen terme (MTM)

- Mon module vectoriel Mnemonic actuel.
- Améliorations :
  - Ajout du champ `importance` (pondération dynamique)
  - Compression d'embeddings selon la récence
  - Filtrage sémantique par tags et score temporel

Bonus : utiliser **sentence-transformers + FAISS GPU** pour booster la vitesse sur gros volumes.

---

3. Mémoire à long terme (LTM)

- Nouvelle couche : **résumés périodiques automatiques.**
- Stocke :
  - Faits stables (“Tom habite à Lyon”)
  - Routines et préférences
  - Relations inter-objets (“Yann⟶ a pour chat ⟶ Anthares”)
- Stockage possible :
  - en JSONL versionné
  - ou en **Neo4j** si tu veux une mémoire *relationnelle* (graphique)

> ⚙️ Mécanisme : tous les X jours → un agent IA lit les logs récents → crée un résumé stable → stocke dans LTM.

---

4. Mémoire épisodique

- Chaque “session” (conversation, journée, événement) devient un **épisode horodaté** :

```
{
  "episode_id": "2025-10-19-matin",
  "summary": "Petit-déj, mise à jour du projet IA",
  "embeddings": [...],
  "tags": ["routine", "développement"]
}
```

- Ces épisodes peuvent être ré-analysés plus tard pour l’auto-apprentissage du modèle.

Phase 2

1. Memory Manager

- Gère l’accès à toutes les couches de mémoire.
- Stratégie :

> - Cherche d’abord dans la STM,
> - Puis interroge Mnemonic (MTM),
> - Puis la LTM si besoin,
> - Combine et résume les résultats avant envoi au modèle.

---

2. Context Composer

- Petit modèle dédié (ex : Mistral 7B ou Phi-4-mini)
- Fusionne plusieurs contextes en un seul bloc cohérent
- Pondère selon pertinence + récence

---

3. Mémoire rétroactive

Après chaque réponse, l’IA s’auto-évalue :

- “Ce que je viens d’apprendre est-il stable ?”
- Si oui → enregistrement dans MTM/LTM.
- Sinon → ignore ou stocke temporairement.

Phase 3

Objectif : Ajouter la perception (vue, son, fichiers).

🔹 Vision

- Utiliser **LLaVA-1.6** ou **Fuyu-8B** pour l’analyse d’images.
- Les embeddings d’image sont aussi indexés dans Mnemonic (type = “vision”).

🔹 Audio

- **Whisper-small** pour la transcription

- Embeddings textuels des résumés ajoutés à la mémoire

🔹 Fichiers

- Analyse contextuelle : extraction des concepts d'un PDF ou texte
- Ajout des concepts clés en mémoire

Phase 4

Objectif : Rendre le système réellement *auto-évolutif*.

🔹 1. Résumés dynamiques

- Un cron/agent périodique relit les conversations récentes pour réévaluer la mémoire :
  - supprime le bruit,
  - renforce les faits récurrents,
  - réécrit certains souvenirs obsolètes.

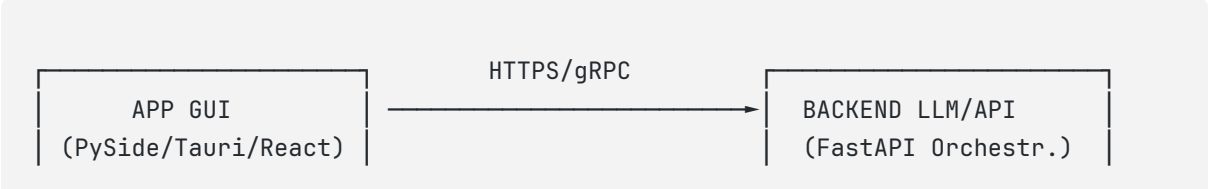
🔹 2. Fine-tuning léger

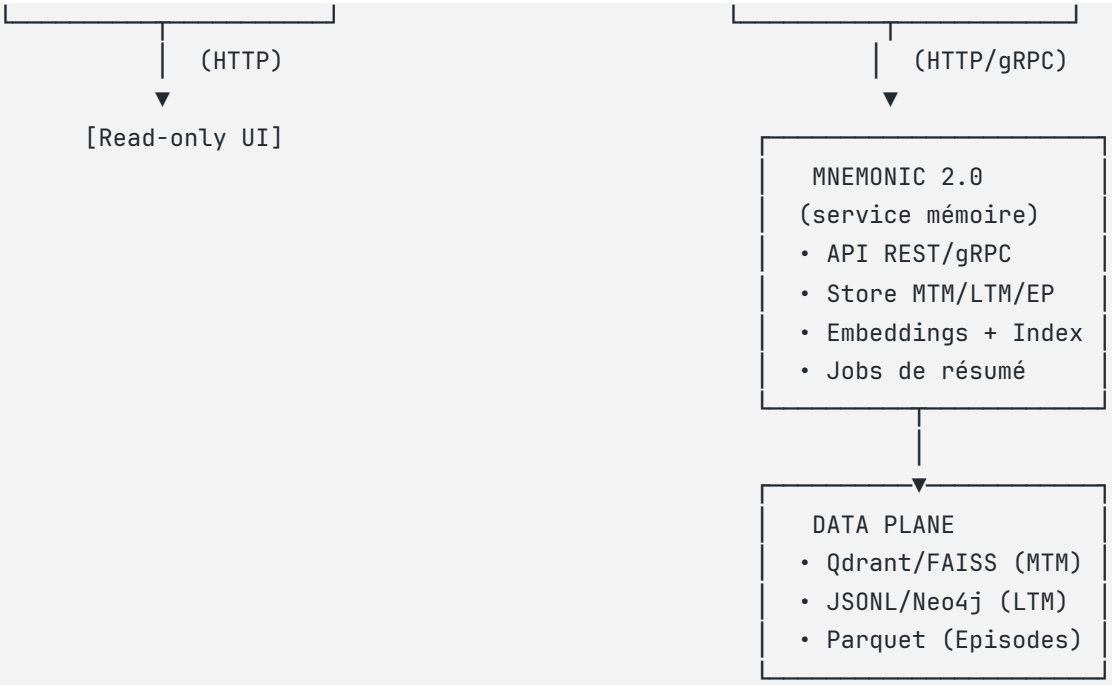
- Stockage des paires `(prompt, réponse idéale, réflexion, réponse finale)`
- Possibilité de faire un fine-tuning local ou LoRA pour personnaliser le modèle.

🔹 3. Auto-étiquetage

- L'IA tague automatiquement les souvenirs : "émotionnel", "logique", "récurrent", "technique"...
- Permet de segmenter et d'optimiser la recherche sémantique.

Fonctionnement (exemple dans l'utilisation de Nexus)





- **GUI** : n'appelle **jamais** directement la mémoire. Elle ne parle qu'au **Backend LLM**.
- **Backend LLM** : orchestre (RAG, vision, outils, domotique), **consomme** Mnemonic via API.
- **Mnemonic** : **service indépendant** (son code, son datastore, son cycle de vie), exposant une
  API
- **contractuelle**.

Contrat d'API de Mnemonic (extrait)

REST (OpenAPI) minimal — contract-first

```
POST   /v1/embeddings/text          # optionnel si Mnemonic calcule en interne
POST   /v1/embeddings/image

POST   /v1/memory/items              # add (STM/MTM/LTM/Episodic)
GET    /v1/memory/items/{id}
POST   /v1/memory/search             # query → top_k + rerank + recency
POST   /v1/memory/summarize          # job de compression vers LTM
POST   /v1/memory/episodes           # créer un épisode
GET    /v1/memory/episodes/{id}
```

```
GET    /v1/stats                    # métriques mémoire
POST   /v1/admin/reindex            # maintenance
```

Modèle de données (simplifié)

```
{
  "id": "uuid",
  "layer": "STM|MTM|LTM|EPISODIC",
  "type": "text|image|file|event|fact",
  "content": { "text": "...", "path": "safe://...", "caption": "..." },
  "embedding": [ ... ],               // présent si MTM
  "meta": {
    "tags": ["home","routine"],
    "importance": 0.73,
    "ts": "2025-10-19T09:42:00Z",
    "episode_id": "2025-10-19-matin",
    "source": "chat|vision|system"
  }
}
```