Quand les IA seront coupées: guide de survie

Séminaire X/STRA, 16 juin 2026

Philippe Helluy

Généralités sur les LLM

Très bref historique

- Les réseaux de neurones artificiels existent depuis **longtemps** (fin des années 50)
- Des hauts et des bas, puis Yann LeCun (reconnaissance d'écriture 1989)
  - Prix Nobel 2024: physique (Hopfield & Hinton, réseaux de neurones), chimie (Hassabis & Jumper, AlphaFold)
- *Attention is all you need* (Vaswani et al., Google, 2017): invention des **transformeurs**
- Sans une énorme **puissance** de calcul, ça ne marcherait pas.

Completion is all you need

- Principe: on se donne un début de texte. Il faut prédire le mot suivant.
- Exemple: le chat mange le … (il faut deviner mulot).
- Mots (ou *tokens*, qui veut dire jeton en anglais):

t_1	t_2	t_3	t_4	t_5	t_6
.	chat	le	mange	matou	mulot

- Corpus: le chat mange le mulot., le matou mange le mulot., etc.

Objectif : Prédire le **prochain token** sachant le contexte (tokens précédents).

Un modèle de langage (*Large Language Model*, LLM) est une fonction f_w qui, à un contexte c, associe une distribution de probabilité sur le prochain token. Les poids (ou paramètres) du modèles sont notés w.

$p = f_w(c)$ où f_w est de $\mathbb{R}^{N_V \times L}$ dans $\mathbb{R}^{N_V}$, N_V: taille du vocabulaire, L: longueur du contexte.

Pour générer un texte entier, on boucle:

1. On prédit le nouveau token.
2. On l'ajoute au contexte c.
3. On recommence.

Le LLM calcule une probabilité pour chaque token du vocabulaire à chaque tour !

Algorithmes de choix du token suivant

Après avoir obtenu la distribution de probabilité sur le vocabulaire, plusieurs stratégies existent pour choisir le token suivant :

- **Algorithme glouton** : on prend toujours le token avec la probabilité la plus élevée. Déterministe, mais peu créatif.
- **Echantillonage aléatoire** : on tire au hasard en tenant compte des probabilités. Introduit de la diversité.
- **Top-k** : on ne considère que les k tokens les plus probables, puis on tire parmi eux.
- **Top-p** : on sélectionne les tokens dont la probabilité cumulée atteint un certain seuil, puis on tire parmi eux.
- **Température T** : permet de lisser la distribution avant le tirage. Si $T = 0$, algorithme glouton, $T = 1$ distribution du LLM, $T > 1$ distribution plus uniforme.

- Codage: à chaque mot (ou *token*) on associe un vecteur à $N_V = 6$ dimensions (*One-Hot Encoding*)

$$. = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{chat} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{le} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{etc.}$$

- Plongement (*embedding*) dans un espace de dimension D plus petite (pour tenir compte des synonymes, entre autres). Par exemple $D = 5$. Le plongement E_{w_0} est une fonction de $\mathbb{R}^{N_V}$ à valeurs dans $\mathbb{R}^D$.
- Chaque token t_i est représenté par un vecteur v_i. Le vecteur w_0 des paramètres du plongement est inconnu: $v_i = E_{w_0}(t_i)$.

- Finalement une phrase est représentée par une matrice de C vecteurs numériques (colonnes) mis côte à côte:

$$c_0 = [v_0 \mid v_1 \mid ... \mid v_{C-1}]$$

  C'est donc un objet dans un espace à $D \times C$ dimensions (par exemple $D = 5$ et $C = 6$, donc 30 dimensions).
- La phrase passe dans L couches de transformeurs T_{w_i}, qui sont des applications de $\mathbb{R}^{D\times C}$ à valeurs dans $\mathbb{R}^{D\times C}$ avec des vecteurs de paramètres inconnus w_i

$$c_i = T_{w_i}(c_{i-1}), \quad i = 1...L.$$

- Quand on s'enfonce dans les couches, la représentation c_i de la phrase initiale est enrichie par les informations des poids w du LLM. La matrice c_L contient l'information extraite par le réseau sur la phrase initiale c_0.

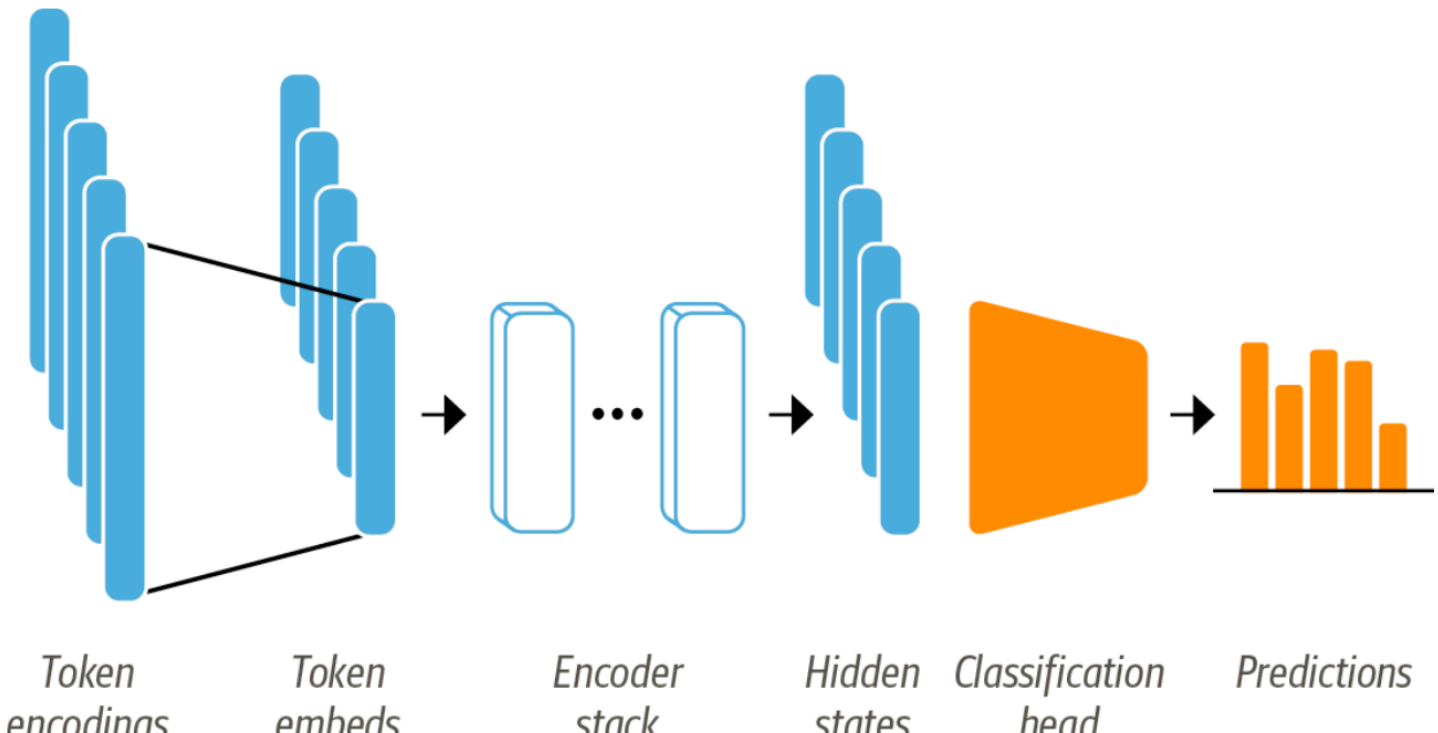
Décodeur

Enfin, le décodeur va permettre de prédire un vecteur de probabilités p dans $\mathbb{R}^{N_V}$: p_i est la probabilité que le mot suivant soit le token i.

$$p = D_{w_{L+1}}(c_L).$$

En résumé (plus de précisions dans [1], [2]):

Décodeur


Token encodings
Token embeds
Encoder stack
Hidden states
Classification head
Predictions

Entrainement

- Choix de la forme des fonctions E_{w_0}, T_{w_i}, $D_{w_{L+1}}$: c'est un compromis entre coût, efficacité, simplicité. C'est un art autant que de la science pour l'instant.
- Historiquement plusieurs formes possibles: CNN, RNN, LSTM, transformeurs. Évolution fortement liée à la puissance de calcul disponible.
- Le vecteur des paramètres w, de taille s est **inconnu**.
- L'entraînement consiste à optimiser le choix de ces paramètres pour que le modèle retrouve au mieux les phrases du corpus.
- C'est la partie la plus difficile du calcul, il faut un super-calculateur, des processeurs spécialement conçus, ça coûte des millions d'euros.
- Ordres de grandeurs pour un LLM "frontière" (en 2026): $C \approx 1M$ (taille contexte), $N_V \approx 100k$ (vocabulaire), $D \approx 10k$ (embedding), $L \approx 100$ (couches), $s \approx 1000$ milliard (paramètres)...

- Une fois que les paramètres w sont calculés, l'inférence est rapide.
- On peut ré-entrainer un réseau pour une tâche spécifique, à coût réduit (*fine tuning*).
- Le *pre-prompt* est essentiel pour obtenir des résultats de qualité.
- Pour des raisons de coût de calcul, un LLM n'apprend pas en temps réel. Sa mémoire à court terme est donc limitée à quelques dizaines de milliers de mots.

Demo: tri de 5 lettres

Je n'ai pas détaillé la forme mathématique précise des fonctions E, T, D car c'est un peu technique.

Plus de détails sur: https://gitlab.math.unistra.fr/helluy/minillm

Visualisation interactive avec: https://bbycroft.net/llm

Apprentissage: SFT et RL

En 2026, un LLM n'est **plus** un "perroquet stochastique". L'entraînement consiste en (au moins) deux étapes:

- Apprentissage supervisé (SL: supervised learning): le LLM apprend à prédire le mot d'après à partir d'un vaste corpus. J'ai trouvé le mot d'après = récompense.
- Apprentissage par renforcement (RL: reinforcement learning): le LLM apprend à générer **tout seul** un texte qui conduit à des résultats corrects (comportement émergent observé: chaînes de pensée). J'ai trouvé le bon résultat = récompense.

C'est l'étape de RL qui rend les LLM très bons en maths.

Aspects politiques: coupures récentes

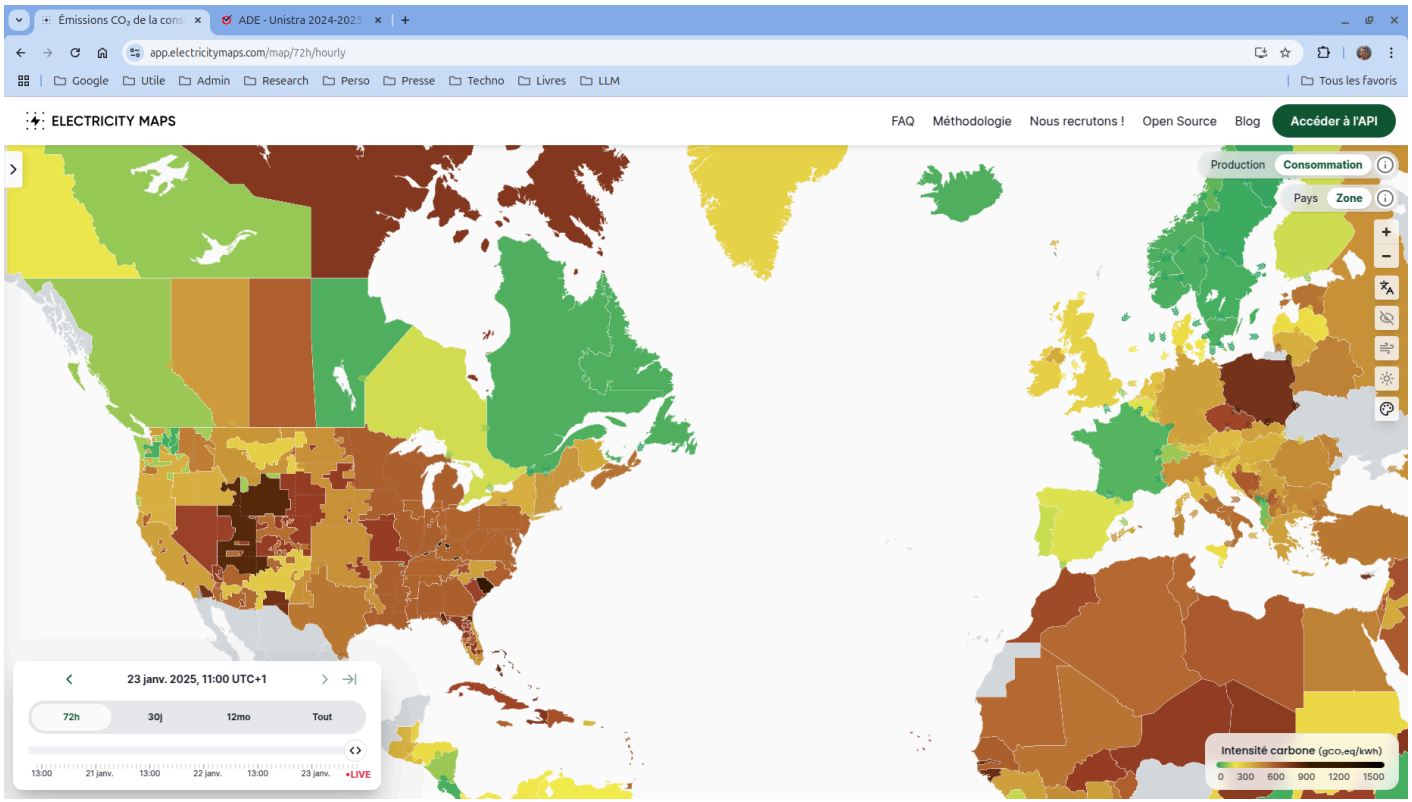
Signaux d'alerte:

- Coupure du modèle Fable d'Anthropic en juin 2026
- Augmentation importante des coûts à prévoir (pour l'instant les fournisseurs d'IA fonctionnent à perte)
- Problèmes de dépendance aux infrastructures cloud étrangères

Objectifs:

- Souveraineté, résilience face aux pannes
- Aspect psychologique: garder le contrôle, se rassurer
- Aspects écologiques: empreinte carbone de la production d'électricité

Empreinte carbone de la production d'électricité



Il vaut mieux faire tourner les serveurs LLM en France, plutôt que dans la plupart des autres pays...

IA Locale

llama.cpp: LLM en local

- Outil pour faire tourner des LLM localement
- Compilation optimisée pour CPU/GPU
- Formats quantifiés: q4, q5, q8 (réduction mémoire du LLM sans perte majeure de qualité)
- Exemple: qwen3.6-35b-q4 sur un portable récent

Installation de llama.cpp

En gros (pour les geeks):

```
git clone https://github.com/ggerganov/llama.cpp
cd llama.cpp
cmake -B build
cmake --build build -j
```

- Téléchargement du modèle sur Hugging Face (ex: https://huggingface.co/unsloth/Qwen3.6-35B-A3B-GGUF)
- Lancement:

```
llama-server -m Qwen3.6-35B-A3B-UD-Q4_K_M.gguf
```

Nécessite environ 24 Go de RAM. On peut aussi ajouter un projecteur multimodal: gestion des images. Il existe des méthodes d'installation plus simples (voir exposé de Morgan)

Optimisations

- Quantification: réduction de la précision des poids (float16 to int4/8)
- KV Cache: mémorisation des états pour accélérer l'inférence
- Attention optimisée: calcul efficace des matrices d'attention
- Flash Attention: optimisation mémoire pour les grandes séquences
- Parallélisation: utilisation multi-thread
- Batch processing: traitement simultané de plusieurs requêtes
- Offloading GPU: utilisation de la mémoire GPU si disponible
- Speculative decoding: génération accélérée
- Turboquant: compression du contexte
- *etc.* (des nouveautés chaque semaine !)

Demo: llama.cpp en chatbot

- Lancement du serveur API:

```
llama-server -m Qwen3.6-35B-A3B-UD-Q4_K_M.gguf
```

- Connexion depuis un client (curl, Python, interface web)
- Test de conversation en local
- Hallucinations possibles, le LLM parle mais n'agit pas.

API, clés et endpoints

- Une API (*Application Programming Interface*) est une interface pour interagir avec un service
- Clé d'API: authentification pour accéder au service
- Point d'entrée (*endpoint*): URL du service (ex: http://localhost:8080/v1/chat/completions)
- Fournisseur (*LLM Providers*): OpenAI, Mistral, Anthropic ou local (llama.cpp)

- MCP (Model Context Protocol): standard pour connecter un LLM à des outils
- Un serveur MCP propose lui aussi un point d'entrée avec une API
- llmtools: https://github.com/phelluy/llmtools
- Actions possibles:
  - Accès Fichiers locaux
  - Exécution de code python
  - Outils externes (recherche web)
  - *etc.*

Demo: Python + SearXNG dans le chatbot

- Intégration comme outil MCP pour le LLM: le preprompt est chargé avec une description des outils MCP.
- Quand le moteur d'inférence (llama-server) détecte un appel d'outil par le LLM, il stoppe l'inférence, exécute l'outil, ajoute la sortie de l'outil dans le prompt, puis relance l'inférence.
- Le LLM peut alors:
  - Rechercher des informations sur le web (outil SearXNG)
  - Écrire et exécuter du code Python
- Alternative à Perplexity, mais en local

Agents LLM en ligne de commande

- Mistral Vibe: assistant IA en CLI
- Outils intégrés: code, math, recherche
- Configuration flexible avec des Skills
- Exécution directe dans le terminal
- Intégration possible avec des modèles locaux

Agents LLM: pourquoi faire ?

- Automatisation de tâches complexes
- Orchestration d'outils multiples
- Prise de décision autonome
- Exemples: correction de copies, analyse de données, génération de rapports

Demo: TP de calcul scientifique, résumé de thèse

- Résolution d'un exercice de calcul scientifique avec Mistral Vibe:
  - Configuration
  - Prompt
  - Vérification des résultats
- Utilisation d'un LLM local pour résumer un document long
  - Technique: découpage du texte + résumé par sections
  - Avantages: pas de fuite de données (tout en local)
  - Alternative à NotebookLM, mais avec contrôle total

Correction de copies "automatique"

- Korektor: logiciel de correction de copies
- Développement assisté par Mistral Vibe
- Fonctionnalités:
  - Scan et reconnaisance d'écriture manuscrite
  - Correction incrémentale et comparative
  - Harmonisation des notes
- IA utilisée: uniquement des LLM servis à l'Unistra !

Demo: logiciel de corrections

- Description du fonctionnement
- Interface

On peut survivre à une panne de l'IA, ouf !

Références

[1] L. Tunstall, L. Von Werra, and T. Wolf, *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[2] V. Vigon, “Cours sur les réseaux de neurones,” 2023.