Comprendre les Grands Modèles de Langage

Chapitre 1 : Fondements et Architecture

Master - Traitement Automatique du Langage Naturel Avancé

Fréjus A. A. Laleye

21 janvier 2026

Plan du cours

Contexte historique

Avant les LLMs :

- Modèles basés sur des règles
- Extraction manuelle de features
- Performances limitées
- Tâches spécifiques

Avec les LLMs :

- Apprentissage de bout en bout
- Features apprises automatiquement
- Performances exceptionnelles
- Modèles généralistes

Exemple

Tâche : Rédiger un email professionnel

Modèles traditionnels :
→ Échec

LLMs (GPT, ChatGPT) :
→ Succès remarquable

Qu'est-ce qu'un LLM ?

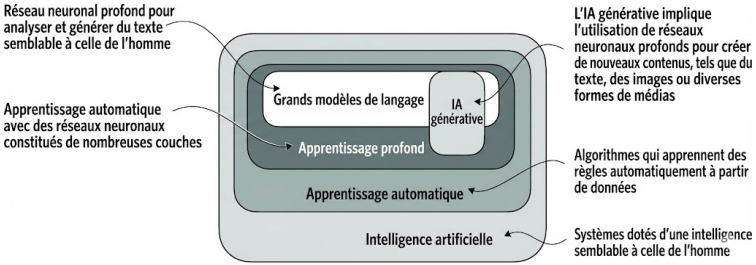
Definition

Un **Grand Modèle de Langage** (Large Language Model - LLM) est un réseau de neurones profond entraîné sur des quantités massives de données textuelles pour comprendre, générer et répondre à du texte de type humain.

Caractéristiques clés

- **Grand :** Des milliards de paramètres ($\theta \in \mathbb{R}^n$, où $n \sim 10^9$ à 10^{12})
- **Données massives :** Entraînement sur des billions de tokens
- **Architecture :** Basé sur le Transformer
- **Objectif :** Modélisation du langage (prédiction du mot suivant)

Hiérarchie des domaines de l'IA



- **IA ⊃ Machine Learning ⊃ Deep Learning ⊃ LLMs**
- Les LLMs sont une application spécialisée du Deep Learning

Modélisation du langage : Formalisme

Objectif fondamental

Étant donné une séquence de tokens $\mathrm{x} = (x_1, x_2, \ldots, x_T)$, modéliser la distribution de probabilité :

$$P(\mathrm{x}) = P(x_1, x_2, \ldots, x_T)$$

Décomposition par la règle de la chaîne :

$$P(\mathrm{x}) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

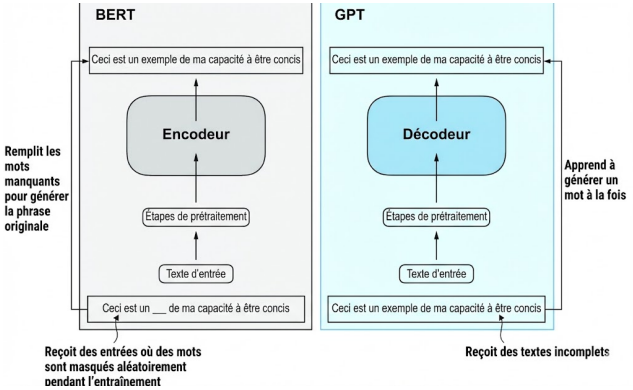
où $x_{<t} = (x_1, \ldots, x_{t-1})$ représente le contexte précédent.

Tâche d'apprentissage

Maximiser la log-vraisemblance :

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t})$$

L'architecture Transformer



Innovation majeure (2017) : « Attention Is All You Need »

- Encodeur : Traite l'entrée complète
- Décodeur : Génère la sortie séquentiellement
- Mécanisme d'attention : Capture les dépendances à longue portée

Mécanisme d'attention : Intuition

Question fondamentale : Comment pondérer l'importance des différents mots dans une séquence ?

Example

« Le chat qui était sur le tapis a mangé la souris. »
Pour comprendre **« a mangé »**, le modèle doit :

- Identifier le sujet : **« Le chat »**
- Ignorer les informations secondaires : « qui était sur le tapis »
- Identifier l'objet : **« la souris »**

$\Rightarrow$ L'attention permet de calculer ces pondérations automatiquement !

Self-Attention : Formalisme mathématique

Entrée : Séquence d'embeddings $X \in \mathbb{R}^{n \times d}$

Étape 1 : Projections linéaires

$$Q = XW^Q \quad \text{(Queries)} \quad W^Q \in \mathbb{R}^{d \times d_k}$$
$$K = XW^K \quad \text{(Keys)} \quad W^K \in \mathbb{R}^{d \times d_k}$$
$$V = XW^V \quad \text{(Values)} \quad W^V \in \mathbb{R}^{d \times d_v}$$

Étape 2 : Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- $QK^\top$: Scores de similarité
- $\sqrt{d_k}$: Facteur de normalisation (stabilité numérique)
- softmax : Conversion en probabilités

Multi-Head Attention

Idée : Apprendre plusieurs représentations d'attention en parallèle

Formulation mathématique

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

où chaque tête est :

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Paramètres :

- h : nombre de têtes (typiquement 8, 12, ou 16)
- $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d_k}$ avec $d_k = d/h$
- $W^O \in \mathbb{R}^{d \times d}$: projection de sortie

Avantage : Chaque tête peut se spécialiser dans différents types de relations

Positional Encoding

Problème : L'attention est invariante à la permutation !

Solution : Ajouter des encodages positionnels aux embeddings

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

où :

- pos : position du token dans la séquence
- i : dimension de l'embedding
- d : dimension totale de l'embedding

Propriétés :

- Valeurs bornées : $\text{PE} \in [-1, 1]$
- Permet au modèle d'apprendre les positions relatives

Architecture complète d'une couche Transformer

Encodeur :

1. Multi-Head Self-Attention
2. Add & Norm (Residual + LayerNorm)
3. Feed-Forward Network
4. Add & Norm

Décodeur :

1. Masked Multi-Head Self-Attention
2. Add & Norm
3. Cross-Attention (avec encodeur)
4. Add & Norm
5. Feed-Forward Network
6. Add & Norm

Feed-Forward Network :

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

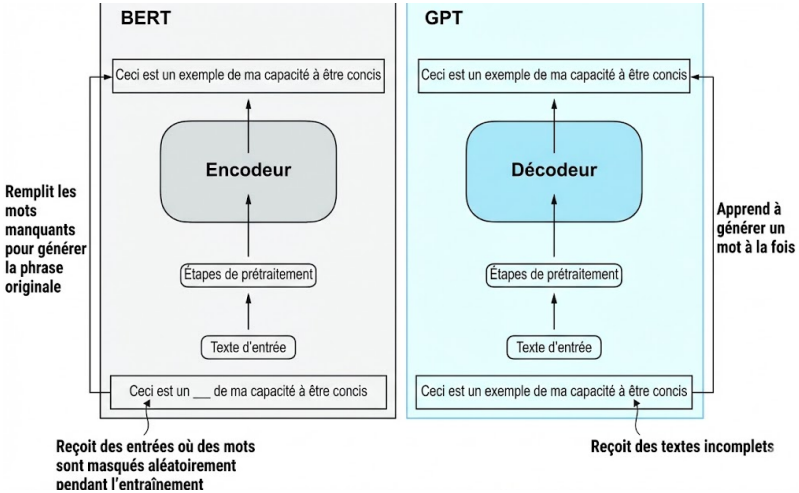
Layer Normalization :

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

Connexions résiduelles :

$$y = x + \text{Sublayer}(x)$$

BERT : Architecture et principe



BERT = **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

BERT : Architecture détaillée

Configuration BERT-Base :

- Nombre de couches : $L = 12$
- Dimension cachée : $H = 768$
- Têtes d'attention : $A = 12$
- Paramètres totaux : ~ 110 millions

Configuration BERT-Large :

- Nombre de couches : $L = 24$
- Dimension cachée : $H = 1024$
- Têtes d'attention : $A = 16$
- Paramètres totaux : ~ 340 millions

Tokens spéciaux :

- `[CLS]` : Token de classification (début de séquence)
- `[SEP]` : Séparateur de phrases

BERT : Masked Language Modeling (MLM)

Objectif : Prédire les tokens masqués en utilisant le contexte bidirectionnel

Procédure de masquage

Pour chaque séquence d'entraînement :

1. Sélectionner aléatoirement 15% des tokens
2. Pour chaque token sélectionné :
   - 80% : remplacer par `[MASK]`
   - 10% : remplacer par un token aléatoire
   - 10% : garder inchangé

Fonction de perte MLM :

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log P(x_i \mid \mathrm{x}_{\setminus \mathcal{M}})$$

où $\mathcal{M}$ est l'ensemble des positions masquées.

BERT : Next Sentence Prediction (NSP)

Objectif : Comprendre les relations entre phrases

Procédure

Créer des paires de phrases (A, B) :

- 50% : B suit réellement A dans le corpus (label = IsNext)
- 50% : B est une phrase aléatoire (label = NotNext)

Format d'entrée :

$$\texttt{[CLS]}\ A_1\ A_2\ \ldots\ A_n\ \texttt{[SEP]}\ B_1\ B_2\ \ldots\ B_m\ \texttt{[SEP]}$$

Fonction de perte NSP :

$$\mathcal{L}_{\text{NSP}} = -\log P(c \mid \mathrm{h}_{\texttt{[CLS]}})$$

où $c \in \{\text{IsNext}, \text{NotNext}\}$ et $\mathrm{h}_{\texttt{[CLS]}}$ est la représentation du token [CLS].

Perte totale BERT :

$$\mathcal{L}_{\text{BERT}} = \mathcal{L}_{\text{MLM}} + \mathcal{L}_{\text{NSP}}$$

BERT : Embeddings d'entrée

Trois types d'embeddings sont sommés :

1. **Token Embeddings** : $E_{token} \in \mathbb{R}^{V \times H}$

$$e_{token} = \text{Lookup}(\text{token\_id})$$

2. **Segment Embeddings** : $E_{seg} \in \mathbb{R}^{2 \times H}$

$$e_{seg} = \begin{cases} E_A & \text{si token} \in \text{phrase A} \\ E_B & \text{si token} \in \text{phrase B} \end{cases}$$

3. **Position Embeddings** : $E_{pos} \in \mathbb{R}^{L_{max} \times H}$

$$e_{pos} = \text{Lookup}(\text{position})$$

Embedding final :

$$e_{input} = e_{token} + e_{seg} + e_{pos}$$

BERT : Fine-tuning pour des tâches spécifiques

Classification de séquence :

$$y = \text{softmax}(\text{W}\text{h}_{\texttt{[CLS]}} + \text{b})$$

Classification de tokens (NER, POS tagging) :

$$y_i = \text{softmax}(\text{W}\text{h}_i + \text{b})$$

Question Answering (SQuAD) :

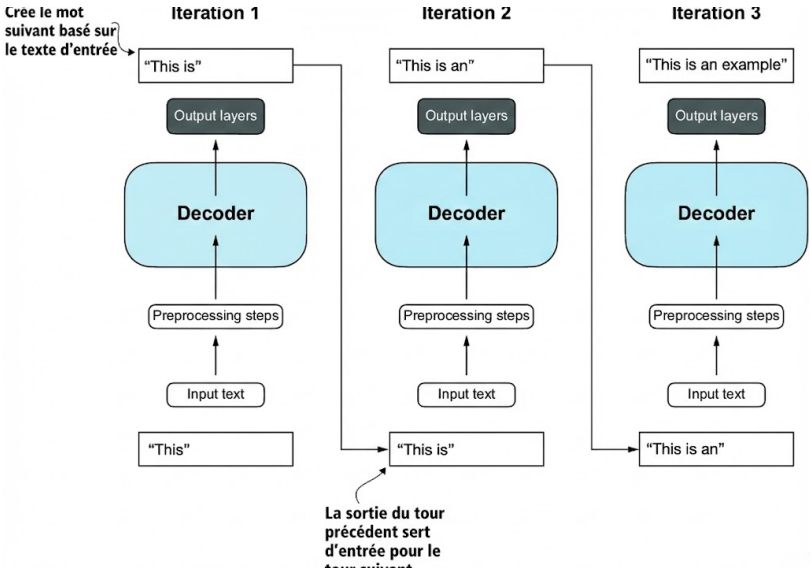
- Prédire les positions de début et fin de la réponse
- Deux vecteurs appris : $\text{S}, \text{E} \in \mathbb{R}^H$

$$P_{\text{start}}(i) = \frac{e^{\text{S}\cdot\text{h}_i}}{\sum_j e^{\text{S}\cdot\text{h}_j}}, \quad P_{\text{end}}(i) = \frac{e^{\text{E}\cdot\text{h}_i}}{\sum_j e^{\text{E}\cdot\text{h}_j}}$$

Avantage

Toutes les couches BERT sont fine-tunées, permettant une adaptation complète à la tâche cible.

GPT : Architecture et principe



GPT : Modélisation autorégressive

Objectif : Maximiser la vraisemblance de la séquence

$$\mathcal{L}_{\text{GPT}} = \sum_{i=1}^{n} \log P(x_i \mid x_1, \ldots, x_{i-1}; \theta)$$

Prédiction du token suivant :

$$\begin{aligned} \mathrm{h}_0 &= \mathrm{X}\mathrm{W}_e + \mathrm{W}_p \\ \mathrm{h}_l &= \text{TransformerBlock}(\mathrm{h}_{l-1}) \quad \forall l \in [1, L] \\ P(x_i) &= \text{softmax}(\mathrm{h}_L^{(i)} \mathrm{W}_e^\top) \end{aligned}$$

où :

- W_e : matrice d'embeddings de tokens
- W_p : matrice d'embeddings positionnels
- L : nombre de couches

GPT : Causal Masking (Masquage causal)

Problème : Empêcher le modèle de « voir le futur »

Solution : Masquer les positions futures dans l'attention

$$\text{Attention}_{\text{causal}}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

où le masque M est défini par :

$$M_{ij} = \begin{cases} 0 & \text{si } i \geq j \\ -\infty & \text{si } i < j \end{cases}$$

Example

Pour la séquence « Le chat mange » :

- « Le » peut voir : « Le »
- « chat » peut voir : « Le », « chat »
- « mange » peut voir : « Le », « chat », « mange »

GPT : Évolution des versions

Modèle	Paramètres	Couches	Dimension
GPT-1	117M	12	768
GPT-2	1.5B	48	1600
GPT-3	175B	96	12288
GPT-4	$\sim$1.7T (estimé)	?	?

Observations :

- Scaling laws : Les performances s'améliorent avec la taille
- Émergence de capacités (few-shot, zero-shot)
- GPT-3 : 300 milliards de tokens d'entraînement

GPT : Génération de texte

Stratégies de décodage :

1. Greedy Decoding :

$$x_t = \arg\max_x P(x \mid x_{<t})$$

2. Beam Search :

- Maintenir les k meilleures séquences
- Score : $\text{score}(\text{x}) = \frac{1}{|\text{x}|^\alpha} \sum_{t=1}^{|\text{x}|} \log P(x_t \mid x_{<t})$

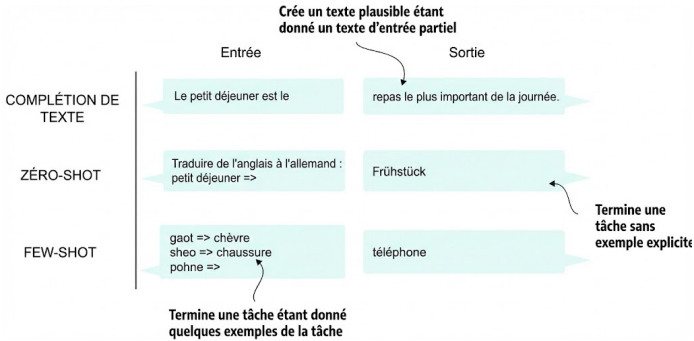
3. Sampling avec température :

$$P_T(x_i) = \frac{\exp(\text{logit}_i / T)}{\sum_j \exp(\text{logit}_j / T)}$$

- $T < 1$: Plus déterministe
- $T > 1$: Plus aléatoire

4. Top-k et Top-p (nucleus) sampling

GPT : Zero-shot et Few-shot Learning



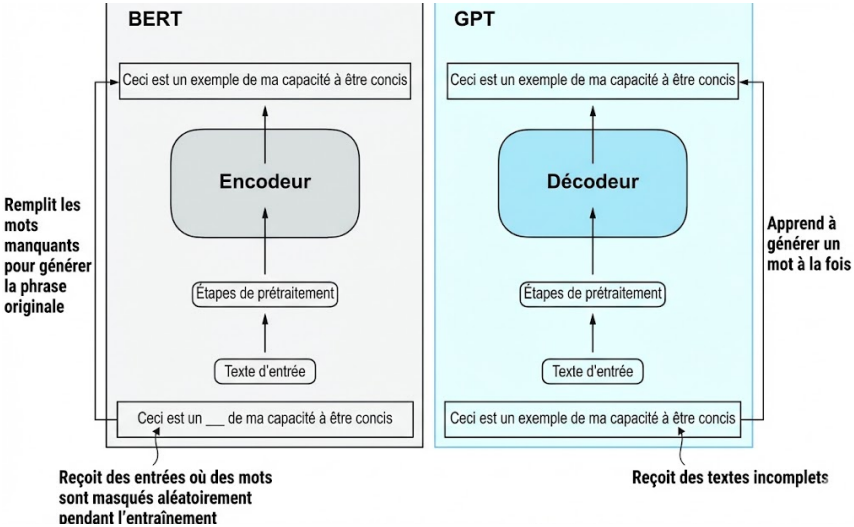
Zero-shot : Résoudre une tâche sans exemple

$$P(y \mid x, \text{task\_description})$$

Few-shot : Apprendre à partir de quelques exemples

$$P(y \mid x, \{(x_1, y_1), \ldots, (x_k, y_k)\})$$

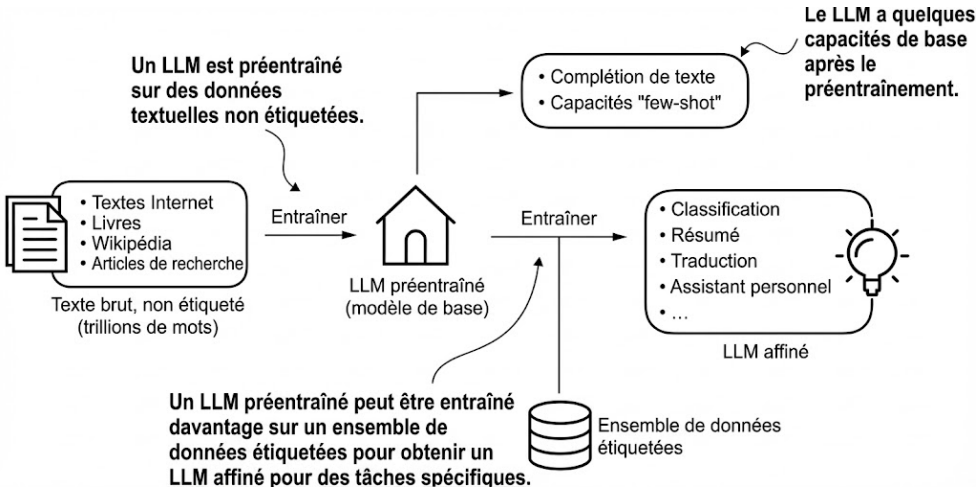
BERT vs GPT : Comparaison architecturale



BERT vs GPT : Tableau comparatif

Caractéristique	**BERT**	**GPT**
Architecture	Encodeur	Décodeur
Attention	Bidirectionnelle	Unidirectionnelle (causale)
Objectif	MLM + NSP	Prédiction suivante
Contexte	Complet	Gauche à droite
Forces		
Classification	✓✓	✓
Compréhension	✓✓	✓
Génération	×	✓✓
Few-shot	×	✓✓
Applications		
NER, POS	✓✓	✓
QA	✓✓	✓
Génération texte	×	✓✓
Dialogue	×	✓✓

Pipeline d'entraînement des LLMs



Pré-entraînement : Données massives

Dataset	Tokens	Proportion	Qualité
CommonCrawl (filtré)	410B	60%	Moyenne
WebText2	19B	22%	Bonne
Books1	12B	8%	Excellente
Books2	55B	8%	Excellente
Wikipedia	3B	3%	Excellente
Total GPT-3	**499B**	**100%**	

Table – Données de pré-entraînement de GPT-3

Prétraitement :

- Filtrage de qualité (perplexité, longueur, etc.)
- Déduplication
- Tokenization (BPE, WordPiece, SentencePiece)

Tokenization : Byte-Pair Encoding (BPE)

Algorithme :

1. Initialiser le vocabulaire avec tous les caractères
2. Répéter jusqu'à atteindre la taille de vocabulaire désirée :
   - Trouver la paire de tokens la plus fréquente
   - Fusionner cette paire en un nouveau token
   - Ajouter au vocabulaire

Example

Corpus : « low », « lower », « lowest »
Évolution :

- Départ : l, o, w, e, r, s, t
- Étape 1 : lo, w, e, r, s, t (fusion de l+o)
- Étape 2 : low, e, r, s, t (fusion de lo+w)
- Étape 3 : low, er, s, t (fusion de e+r)

Fine-tuning : Adaptation aux tâches

Deux approches principales :

1. Instruction Fine-tuning

- Dataset : Paires (instruction, réponse)
- Exemple : « Traduire en français : Hello » → « Bonjour »
- Modèles : InstructGPT, FLAN-T5

2. Classification Fine-tuning

- Dataset : Paires (texte, label)
- Exemple : « Ce film est génial ! » → Positif
- Ajouter une couche de classification

Fonction de perte pour classification :

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \hat{y}_{ic}$$

RLHF : Reinforcement Learning from Human Feedback

Motivation : Aligner les LLMs avec les préférences humaines

Pipeline en 3 étapes :

1. **Supervised Fine-tuning (SFT)**
   - Entraîner sur des démonstrations humaines de haute qualité
2. **Reward Model Training**
   - Collecter des comparaisons humaines : $y_1 \succ y_2$
   - Entraîner un modèle de récompense : $r_\phi(x, y)$
3. **RL Optimization (PPO)**
   - Maximiser : $\mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta}[r_\phi(x, y)] - \beta \mathrm{KL}(\pi_\theta \| \pi_{\text{ref}})$

Résultat : ChatGPT, Claude, etc.

Métriques pour la modélisation du langage

1. Perplexité (Perplexity)

$$\text{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(x_i \mid x_{<i})\right)$$

- Mesure la « surprise » du modèle
- Plus bas = meilleur
- Interprétation : nombre moyen de choix équiprobables

2. Bits per Character (BPC)

$$\text{BPC} = -\frac{1}{N\log 2}\sum_{i=1}^{N}\log P(x_i \mid x_{<i})$$

- Mesure l'efficacité de compression
- Indépendant de la taille du vocabulaire

Métriques pour les tâches en aval

Classification :

- Accuracy : $\frac{\text{Correct}}{\text{Total}}$
- F1-Score : $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

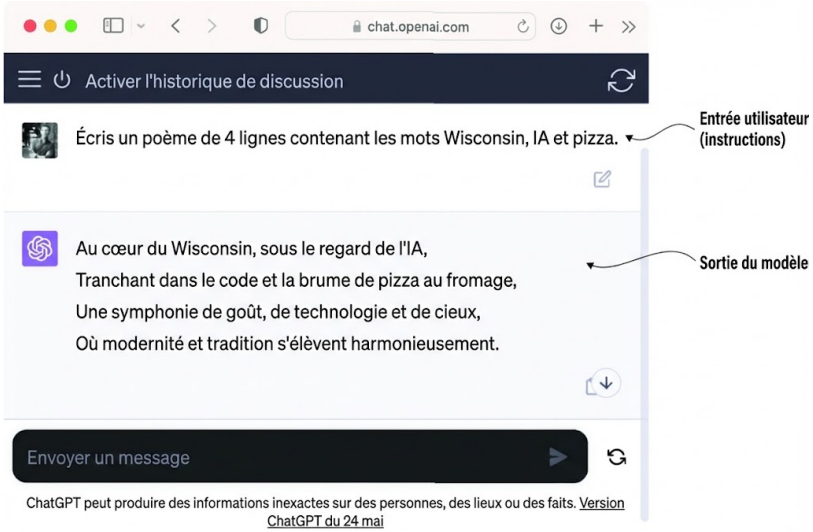
Question Answering :

- Exact Match (EM) : Correspondance exacte
- F1 : Chevauchement de tokens

Génération de texte :

- BLEU : $\text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$
- ROUGE : Rappel de n-grammes
- BERTScore : Similarité sémantique basée sur BERT

Applications des LLMs



Domaines d'application

NLP Classique :

- Classification de texte
- Reconnaissance d'entités (NER)
- Analyse de sentiments
- Question-Answering
- Résumé automatique
- Traduction

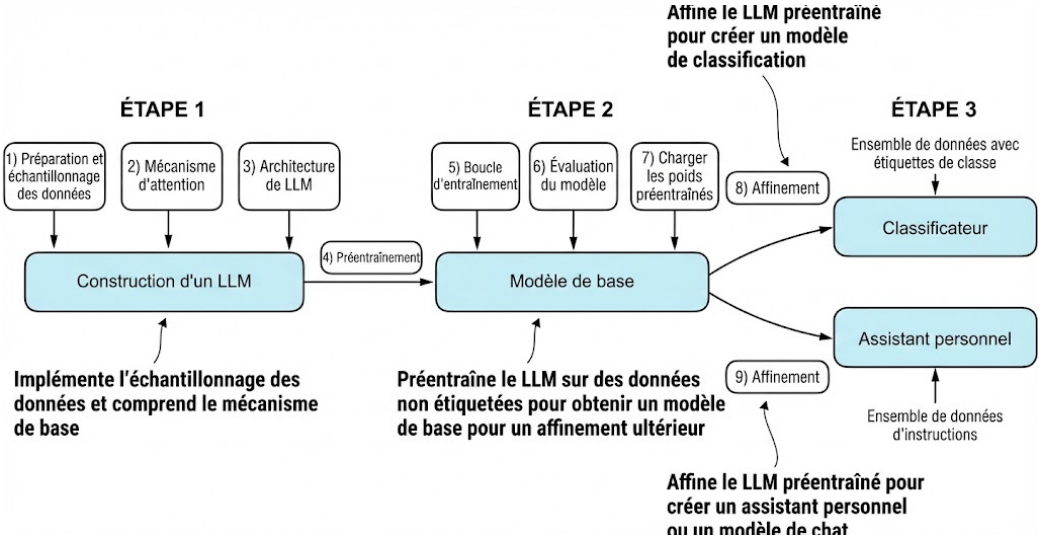
Applications émergentes :

- Assistants conversationnels
- Génération de code
- Rédaction créative
- Analyse de documents
- Recherche d'information
- Tuteurs intelligents

Domaines spécialisés

- Médecine : Med-PaLM, BioGPT
- Droit : LegalBERT
- Finance : FinBERT, BloombergGPT
- Science : Galactica, SciBERT

Les 3 étapes de construction



Étape 1 : Architecture et préparation des données

Choix architecturaux :

- Nombre de couches L
- Dimension cachée d_{model}
- Nombre de têtes d'attention h
- Taille du vocabulaire V
- Longueur de contexte maximale n_{ctx}

Estimation du nombre de paramètres :

$$\text{Params} \approx 12 \cdot L \cdot d_{\text{model}}^2 + V \cdot d_{\text{model}}$$

Préparation des données :

- Collecte et nettoyage
- Tokenization
- Création des batches

Étape 2 : Pré-entraînement

Configuration d'entraînement :

- Optimiseur : Adam avec $\beta_1 = 0.9, \beta_2 = 0.95$
- Learning rate : Warmup puis décroissance cosine
- Batch size : Aussi grand que possible (gradient accumulation)
- Gradient clipping : $\|\nabla\| \leq 1.0$

Learning rate schedule :

$$\text{lr}(t) = \begin{cases} \text{lr}_{\max} \cdot \frac{t}{T_{\text{warmup}}} & \text{si } t < T_{\text{warmup}} \\ \text{lr}_{\min} + \frac{1}{2}(\text{lr}_{\max} - \text{lr}_{\min}) \left(1 + \cos\left(\frac{t - T_{\text{warmup}}}{T_{\max} - T_{\text{warmup}}} \pi\right)\right) & \text{sinon} \end{cases}$$

Coût computationnel :

$$C \approx 6 \cdot N \cdot D$$

où N = nombre de paramètres, D = nombre de tokens

Étape 3 : Fine-tuning et déploiement

Stratégies de fine-tuning :

1. Full Fine-tuning

- Tous les paramètres sont mis à jour
- Meilleure performance
- Coûteux en mémoire

2. Parameter-Efficient Fine-Tuning (PEFT)

- **LoRA** : Low-Rank Adaptation

$$W' = W + BA, \quad B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times d}, r \ll d$$

- **Adapter Layers** : Petits modules insérés entre les couches
- **Prefix Tuning** : Optimiser uniquement les embeddings de préfixe

Avantages PEFT :

- Moins de mémoire (1-10% des paramètres)

Défis techniques

1. Coût computationnel

- GPT-3 : $\sim$\$4.6M pour l'entraînement
- Nécessite des clusters GPU/TPU massifs
- Empreinte carbone importante

2. Longueur de contexte limitée

- Complexité quadratique : $\mathcal{O}(n^2)$
- Solutions : Sparse attention, Linear attention, Sliding window

3. Hallucinations

- Génération d'informations factuellement incorrectes
- Difficulté à distinguer faits et fiction

4. Biais et équité

- Reproduction des biais présents dans les données

Limitations théoriques

1. Compréhension vs Mémorisation

- Les LLMs « comprennent »-ils vraiment ?
- Débat philosophique sur la conscience et la compréhension
- Test de Turing et au-delà

2. Raisonnement et logique

- Difficultés avec le raisonnement multi-étapes
- Erreurs de calcul arithmétique
- Solutions : Chain-of-Thought prompting, Tool use

3. Connaissances temporelles

- Connaissances figées au moment de l'entraînement
- Pas de mise à jour en temps réel
- Solutions : Retrieval-Augmented Generation (RAG)

Directions de recherche

1. Efficacité et scalabilité

- Mixture of Experts (MoE)
- Quantization et compression
- Efficient attention mechanisms

2. Multimodalité

- Vision + Language : CLIP, Flamingo, GPT-4V
- Audio + Language : Whisper, AudioLM
- Modèles unifiés : Gemini, GPT-4o

3. Agents et outils

- LLMs comme contrôleurs d'agents
- Utilisation d'outils externes (calculatrice, API, etc.)
- AutoGPT, BabyAGI, LangChain

Scaling Laws

Observation empirique (Kaplan et al., 2020) :

La perte suit une loi de puissance en fonction de :

- Nombre de paramètres N
- Taille du dataset D
- Compute C

$$L(N, D, C) \approx \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{D_c}{D}\right)^{\alpha_D} + \left(\frac{C_c}{C}\right)^{\alpha_C}$$

Implications :

- Performances prédictibles avec l'échelle
- Allocation optimale des ressources
- Chinchilla scaling laws : Équilibrer N et D

Règle Chinchilla

Résumé du chapitre

Points clés :

1. Les LLMs sont des réseaux de neurones profonds basés sur le Transformer
2. **BERT** : Encodeur bidirectionnel, excellent pour la compréhension
3. **GPT** : Décodeur autorégressif, excellent pour la génération
4. Le pré-entraînement sur des données massives est crucial
5. Le fine-tuning permet l'adaptation à des tâches spécifiques
6. Les capacités émergentes apparaissent avec l'échelle

Formalismes essentiels :

- Self-attention : $\text{Attention}(\mathrm{Q}, \mathrm{K}, \mathrm{V}) = \text{softmax}\left(\frac{\mathrm{Q}\mathrm{K}^\top}{\sqrt{d_k}}\right)\mathrm{V}$
- Modélisation du langage : $P(\mathrm{x}) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$
- Causal masking pour GPT
- MLM et NSP pour BERT

Questions de réflexion

1. Pourquoi le mécanisme d'attention est-il plus efficace que les RNNs pour capturer les dépendances à longue portée ?
2. Quels sont les avantages et inconvénients de l'approche bidirectionnelle (BERT) vs unidirectionnelle (GPT) ?
3. Comment le masquage causal dans GPT garantit-il que le modèle ne « triche » pas pendant l'entraînement ?
4. Pourquoi les LLMs développent-ils des capacités émergentes (few-shot, zero-shot) sans entraînement explicite ?
5. Quelles sont les implications éthiques de l'utilisation des LLMs à grande échelle ?

Ressources pour aller plus loin

Articles fondateurs :

- Vaswani et al. (2017) : « Attention Is All You Need »
- Devlin et al. (2018) : « BERT : Pre-training of Deep Bidirectional Transformers »
- Radford et al. (2018, 2019) : « GPT, GPT-2 »
- Brown et al. (2020) : « Language Models are Few-Shot Learners » (GPT-3)

Ressources en ligne :

- The Illustrated Transformer (Jay Alammar)
- Hugging Face Transformers Library
- Stanford CS224N : Natural Language Processing with Deep Learning
- Papers with Code : State-of-the-art benchmarks

Implémentations :

- PyTorch, TensorFlow
- Hugging Face Transformers

Exercices pratiques

Exercice 1 : Calcul d'attention
Calculer manuellement l'attention pour une séquence de 3 tokens avec $d_k = 2$.

Exercice 2 : Implémentation
Implémenter en PyTorch :

- Une couche de self-attention
- Le masquage causal pour GPT
- Le masquage MLM pour BERT

Exercice 3 : Fine-tuning
Fine-tuner un modèle BERT pré-entraîné sur une tâche de classification de sentiments.

Exercice 4 : Analyse
Analyser les patterns d'attention appris par un modèle pré-entraîné sur différents types de phrases.

Merci pour votre attention !

Questions ?

Contact : [votre.email@universite.fr]