Traitement du Langage Naturel (NLP)
Des fondamentaux aux modèles de langage avancés

Florian Valade

Université Gustave Eiffel

14 mars 2025

L'écosystème des entreprises d'IA

OpenAI (ChatGPT, GPT-4)

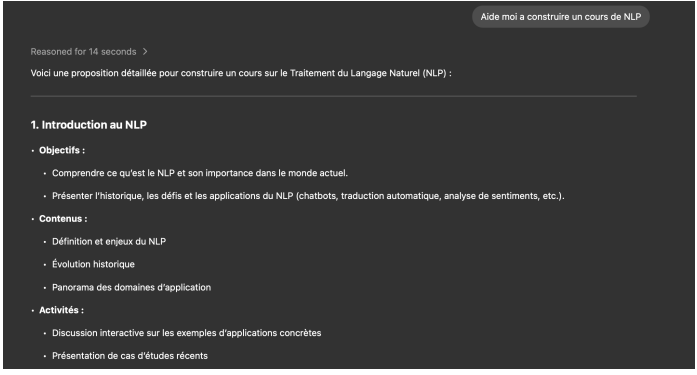
Mistral AI

Meta (LLaMA)

Anthropic (Claude)

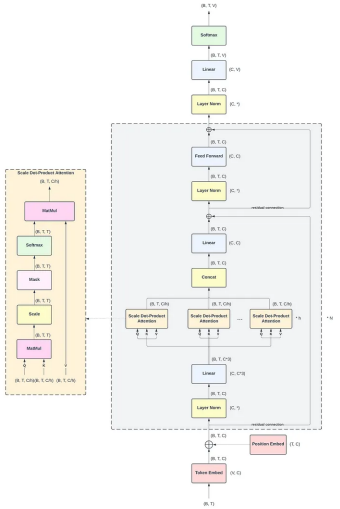
Google (Gemini)

Interface utilisateur des LLMs : ChatGPT



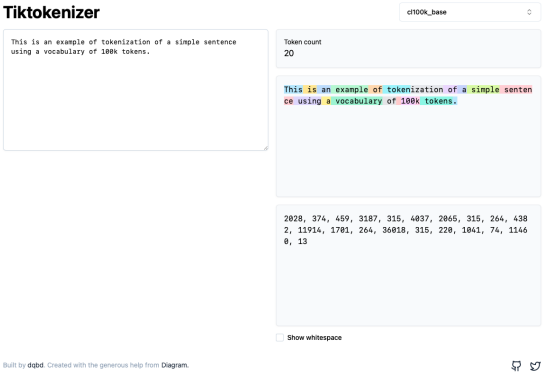
- Interface conversationnelle intuitive
- Capacité à comprendre le contexte et les instructions
- Génération de réponses cohérentes et informatives
- Adaptabilité à différents types de requêtes

Ce qu'il y a derrière l'interface : Architecture des LLMs



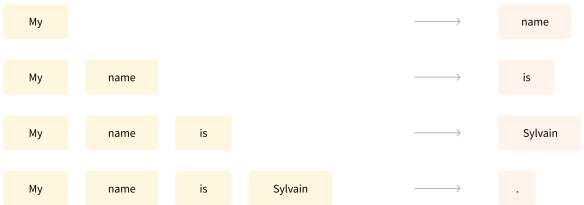
- Architecture basée sur les Transformers
- Milliards de paramètres entraînés sur d'énormes corpus de texte
- Mécanisme d'attention pour capturer les dépendances à longue distance
- Apprentissage auto-supervisé (prédiction du token suivant)

Rappel : La tokenisation



- Les modèles de langage ne comprennent pas directement le texte
- Le texte est converti en séquences de tokens (unités de base)
- Différentes méthodes : par caractère, par mot, par sous-mot (BPE)
- Les tokens sont ensuite convertis en IDs numériques pour le modèle

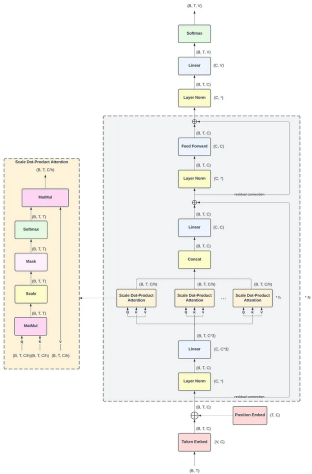
Comment fonctionnent les LLMs : Génération de texte



- Génération auto-régressive : un token à la fois
- À chaque étape, le modèle prédit le token le plus probable suivant
- Le token prédit est ajouté au contexte pour la prédiction suivante
- Processus répété jusqu'à obtenir la réponse complète

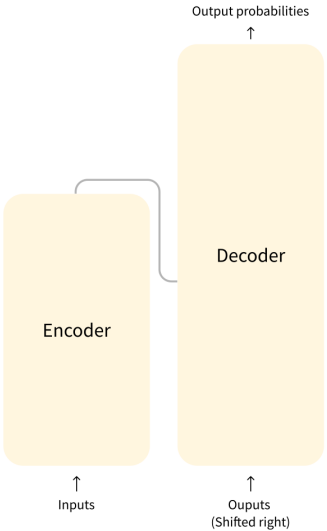
Architecture révolutionnaire en NLP

Architecture détaillée des LLMs : Le modèle GPT



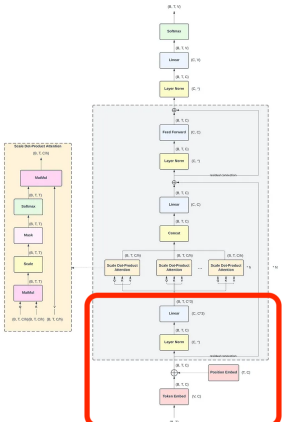
- **Embedding** : conversion des tokens en vecteurs denses
- **Positional encoding** : intégration de l'information de position
- **N blocs Transformer** : traitement contextuel profond
- **Layer normalization** : stabilise l'apprentissage
- **Connexions résiduelles** : facilite l'entraînement de réseaux profonds
- **Tête de classification** : prédit le token suivant

Architecture originale des Transformers : Encoder-Decoder



- Architecture présentée dans *"Attention is All You Need"* (2017)
- **Encoder** : traite l'ensemble de la séquence d'entrée
- **Decoder** : génère la séquence de sortie token par token
- Parfaite pour la **traduction automatique**, où la taille de sortie est inconnue
- GPT utilise uniquement la partie **decoder** (auto-régressive)
- BERT utilise uniquement la partie **encoder** (bi-directionnelle)

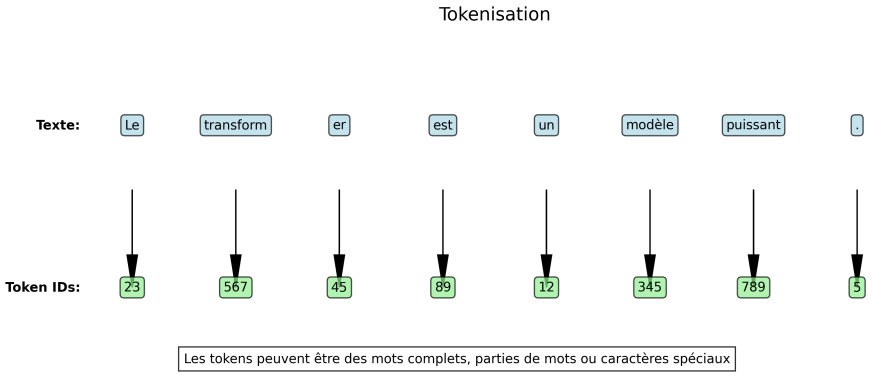
Focus sur l'entrée du modèle GPT



Plusieurs étapes :

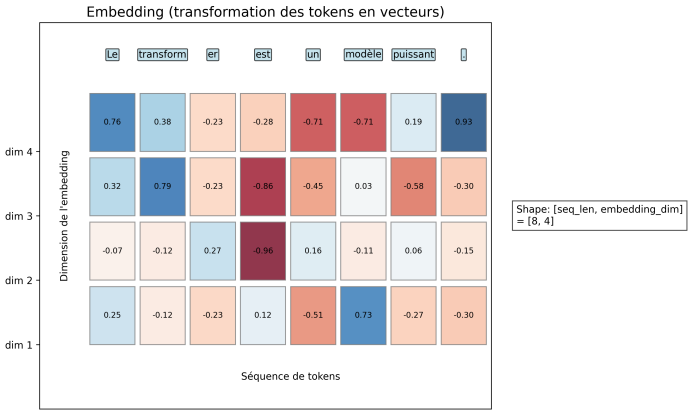
- Tokenisation : Texte $\rightarrow$ Tokens $[context\_length]$
- Embedding : Tokens $\rightarrow$ Vecteurs $[context\_length, embedding\_dim]$
- Positional encoding : Vecteurs $\rightarrow$ Vecteurs $[context\_length, embedding\_dim]$
- Layer normalization : Vecteurs $\rightarrow$ Vecteurs $[context\_length, embedding\_dim]$
- Projections QKV : Vecteurs $\rightarrow$ Vecteurs $[context\_length, 3 \times embedding\_dim]$

Du texte aux vecteurs : Tokenisation



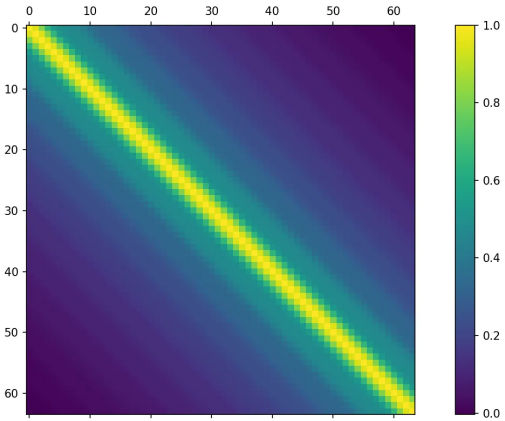
- **Tokenisation** : découpage du texte en unités (tokens) que le modèle peut traiter
- Différentes méthodes : par caractère, par mot, par sous-mot (BPE)
- Les tokens sont ensuite convertis en IDs numériques pour le modèle

Embedding : Conversion des tokens en vecteurs



- **Embedding** : transformation des tokens en vecteurs de nombres réels
- Dimension d'embedding typique : 768 à 4096 selon la taille du modèle
- Les vecteurs capturent les relations sémantiques entre les tokens
- La matrice d'embedding est apprise lors de l'entraînement du modèle

Positional Encoding : Intégrer l'information de position



- Les Transformers n'ont pas de notion intrinsèque de **position**
- Nécessité d'ajouter cette information explicitement
- Encodage sinusoïdal (Vaswani et al.) :

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d})$$

- **Propriétés** :
  - Fréquences différentes pour chaque dimension
  - Valeurs bornées entre $[-1, 1]$
  - Permet de généraliser à des positions non vues

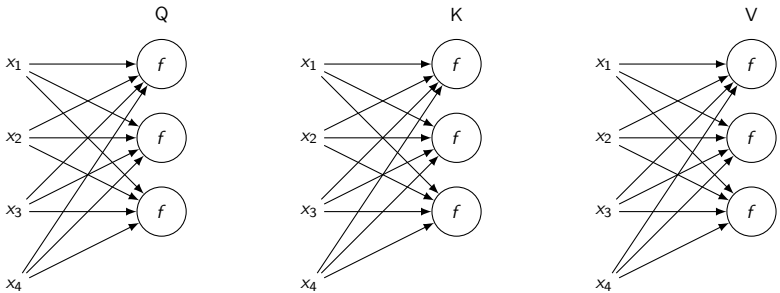
Layer Normalization : Stabiliser l'apprentissage

$$\hat{x} = \frac{x - \mu}{\sigma}$$
$$y = \gamma \cdot \hat{x} + \beta$$

- μ et σ sont les moyennes et écarts-types des vecteurs de token
- Normalise chaque vecteur de token indépendamment
- Réduit la covariance interne (shift covariate)
- Stabilise et accélère l'entraînement des réseaux profonds
- Paramètres apprenables γ et β pour préserver la capacité expressive

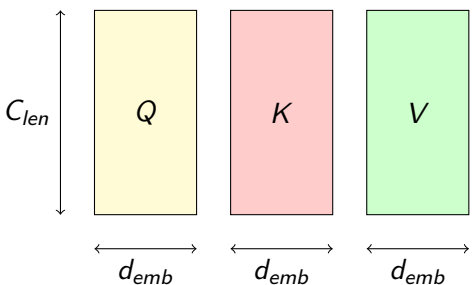
Le cœur du Transformer : L'attention

Projections linéaires pour Query (Q), Key (K), Value (V)



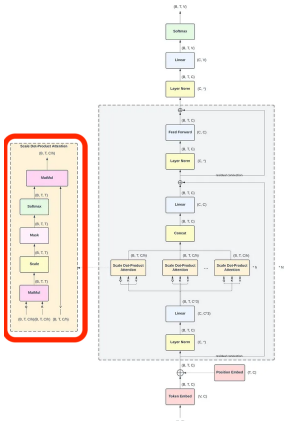
- **Query (Q)**, **Key (K)**, **Value (V)** : projections différentes du même input
- Score d'attention : $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- Permet de capturer différents types de relations entre tokens
- Query représente la question, Key représente les réponses possibles, Value représente les réponses

Récapitulation : Les matrices Query, Key et Value



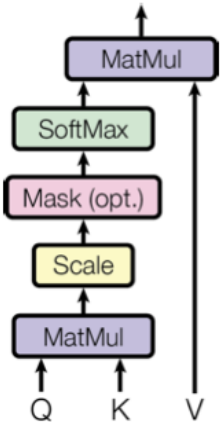
- Trois matrices de taille [contexte, embedding] ou une matrice de taille [contexte, 3 $\times$ embedding]
- **Queries (Q)** : Représentent ce que recherche chaque token
  - "Quelles informations sont importantes pour moi ?"
- **Keys (K)** : Encapsulent les informations disponibles dans chaque token
  - "Voici les informations que je peux fournir"
- **Values (V)** : Contiennent les informations effectives à combiner
  - "Voici mon contenu qui sera utilisé pour la sortie"
- Les scores d'attention (QK^T) déterminent quelles valeurs (V) sont pertinentes

Le mécanisme d'attention : Comprendre les relations contextuelles



- L'attention est le **cœur des Transformers**
- Permet de **pondérer dynamiquement** l'importance de chaque token
- Capture les **dépendances à longue distance**
- Chaque token peut "prêter attention" à tous les tokens précédents

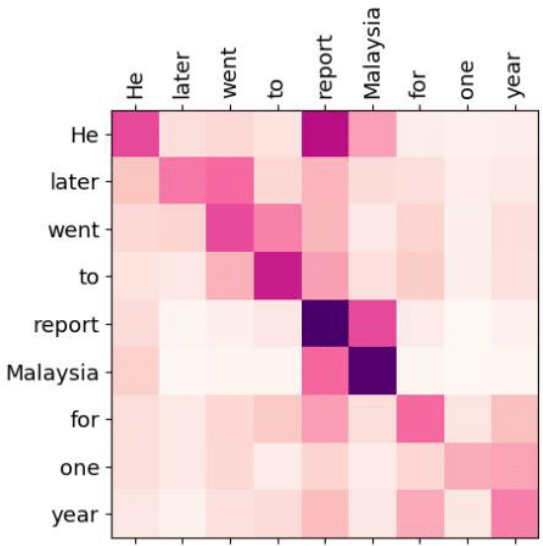
Scaled Dot-Product Attention



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

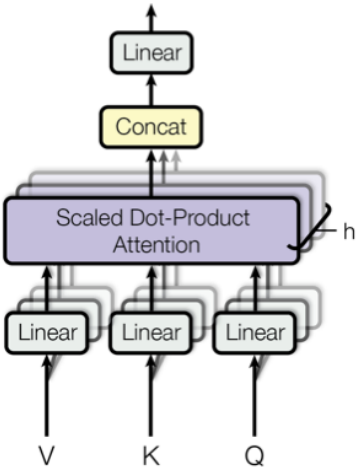
- QK^T calcule les scores de similarité [contexte $\times$ contexte]
- Division par $\sqrt{d_k}$ stabilise les gradients
- **softmax** transforme les scores en poids [0,1]
- Multiplication par V produit une moyenne pondérée des valeurs

Exemple de matrice d'attention



- Visualisation des scores d'attention entre tokens
- Chaque ligne représente l'attention d'un token vers tous les autres
- Les zones plus foncées indiquent une attention plus forte
- Permet d'interpréter quelles parties du texte sont liées
- Révèle les relations grammaticales et sémantiques capturées

Multi-Head Attention : Attention parallèle à plusieurs niveaux

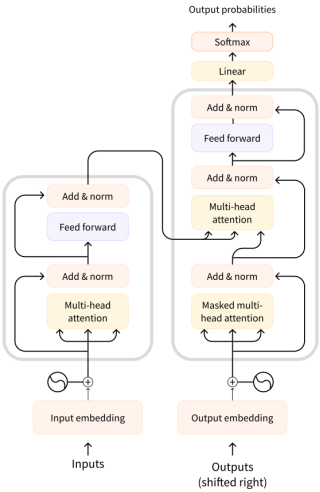


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

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

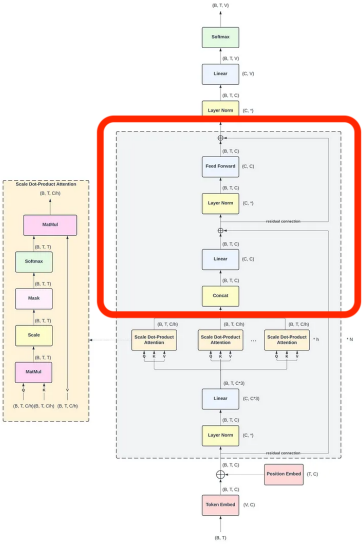
- Plusieurs "têtes" d'attention en parallèle
- Chaque tête se spécialise sur des aspects différents
  - Syntaxe, sémantique, coréférence, etc.
- GPT-2 utilise 12 têtes, GPT-3 jusqu'à 96 têtes
- Les sorties des têtes sont concaténées puis projetées
- Permet de capturer plusieurs types de relations simultanément

Architecture du Transformer : Le modèle original encoder-decoder



- **Architecture encoder-decoder** :
  - **Encoder** : Traite le texte source
  - **Decoder** : Génère le texte cible
- **Composants clés** :
  - Multi-head attention
  - Feed-forward networks
  - Residual connections
  - Layer normalization
- **Caractéristiques** :
  - Traitement parallèle des tokens
  - Attention bidirectionnelle (encoder)
  - Attention masquée (decoder)
  - Base de tous les modèles modernes

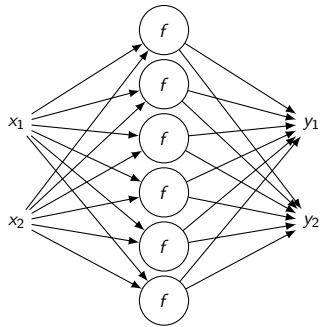
Après l'attention : Feed-Forward Network



- Après le mécanisme d'attention, vient un réseau feed-forward
- Appliqué indépendamment à chaque position (token)
- Composé de deux transformations linéaires et d'une activation
- Permet de transformer les représentations contextualisées
- Augmente la capacité de modélisation non-linéaire du réseau
- Contient la majorité des paramètres du Transformer

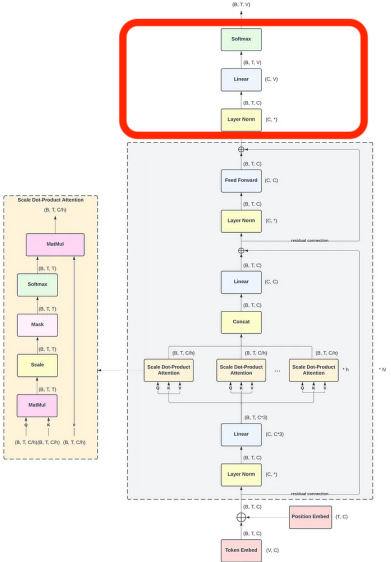
Structure du MLP dans les Transformers

Projection linéaire dans le Feed-Forward Network



- **Expansion** : La dimension cachée est 4 fois plus grande que l'entrée/sortie
- **GELU** : Fonction d'activation non-linéaire plus performante que ReLU
- **Projection** : Retour à la dimension d'origine
- **Dropout** : Régularisation pour éviter le surapprentissage

La tête de classification : Structure



- **Dernière couche** du modèle LLM
- Transforme les représentations contextuelles en **probabilités** sur le vocabulaire
- Structure simple : **couche linéaire** suivie d'un **softmax**
- La dimension de sortie correspond à la **taille du vocabulaire** (50K-100K tokens)
- Dans GPT, la matrice de poids est souvent **partagée** avec la matrice d'embedding (weight tying)

La tête de classification : Prédiction du token suivant

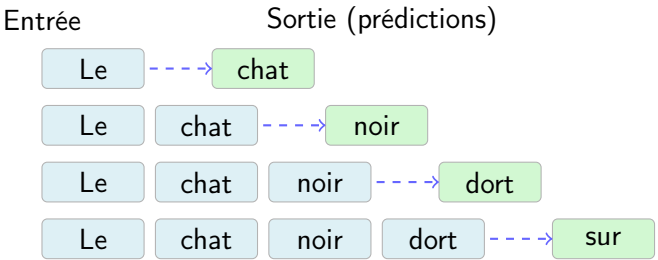
- **Adaptable** selon la tâche :
  - Génération de texte : prédiction du token suivant
  - Classification : prédiction d'une classe
  - Question-réponse : extraction de réponses

Fonction Softmax : Conversion des logits en probabilités

$$P(token_i|contexte) = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{|V|} e^{z_j}}$$

où $z_i = (Wx + b)_i$ sont les logits pour le token i et $|V|$ est la taille du vocabulaire

Format de sortie et masquage causal



Masque d'attention

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

- Le modèle génère des prédictions pour **tous les tokens** simultanément
- Pour chaque position, on prédit le token **suivant**
- Un **masque causal** empêche chaque token de "voir le futur" :
  - Le premier token ne voit que lui-même
  - Le deuxième voit le premier et lui-même
  - Etc.
- La sortie est de la **même dimension** que l'entrée mais **décalée** d'une position

Comment générer du texte à partir du vecteur de probabilités ?

Vecteur de probabilités sur le vocabulaire

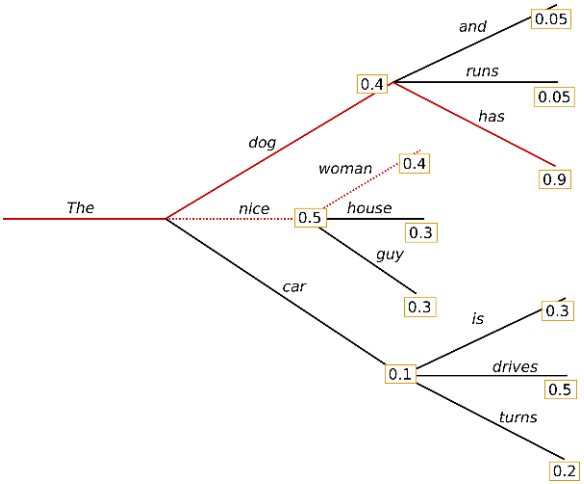
Stratégies déterministes

- Greedy Search
- Beam Search

Stratégies stochastiques

- Sampling
- Top-K sampling
- Top-P sampling (Nucleus)

Greedy Search : La méthode la plus simple

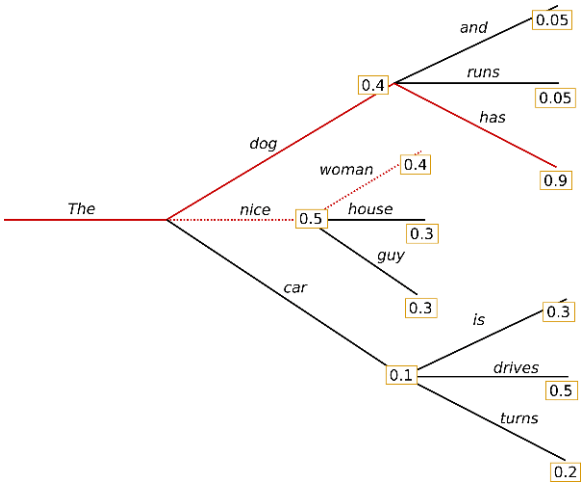


$$w_t = \underset{w}{\text{argmax}}\ p(w|w_{1:t-1})$$

- À chaque étape, choisir le token le **plus probable**
- Méthode **déterministe** : pour une entrée donnée, toujours la même sortie
- **Inconvénients** :
  - Pas de diversité
  - Tendance à se répéter
  - Peut ignorer des chemins globalement meilleurs

Illustrations adaptées de Hugging Face : `https://huggingface.co/blog/how-to-generate`

Beam Search : Explorer plusieurs chemins en parallèle



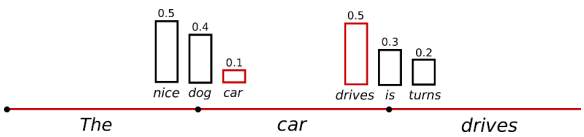
- Maintient les **k meilleures hypothèses** à chaque étape (beam width)
- Pour chacune, explore toutes les possibilités au pas suivant
- Sélectionne les **k meilleures** parmi les **k** × **V** combinaisons
- Score d'une séquence $Y = (y_1, ..., y_t)$:

$$score(Y) = \sum_{i=1}^{t} \log p(y_i | y_{1:i-1})$$

- **Problèmes** :
  - Toujours déterministe
  - Peu de diversité (les chemins convergent)
  - Coûteux pour de grandes valeurs de k

Sampling : Introduire de l'aléatoire dans la génération

$$w_t \sim p(w|w_{1:t-1})$$

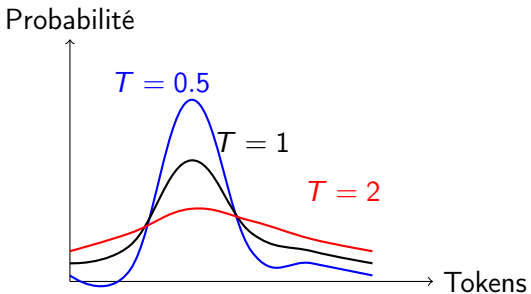


- Échantillonne un token selon la distribution de probabilité
- **Avantages** :
  - Génère des textes **variés**
  - Peut trouver des formulations créatives
  - Plus naturel pour la conversation
- **Inconvénients** :
  - Peut générer des tokens improbables

Température : Contrôler l'aléatoire dans le sampling

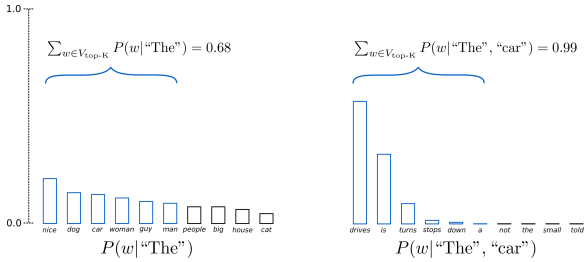
$$p_T(w_i|w_{1:t-1}) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

où z_i sont les logits pour le token i



- Contrôle la "netteté" de la distribution
- **Température basse** ($T < 1$) :
  - Plus conservateur
  - Accentue les différences
  - Proche du greedy search
- **Température haute** ($T > 1$) :
  - Plus exploratoire
  - Aplatit la distribution
  - Génération plus diverse
- $T = 0$: équivalent à greedy
- Hyperparamètre crucial pour la qualité de génération

Top-K Sampling : Limiter l'espace des possibles



1. Sélectionner les **K tokens** les plus probables
2. **Renormaliser** leurs probabilités :

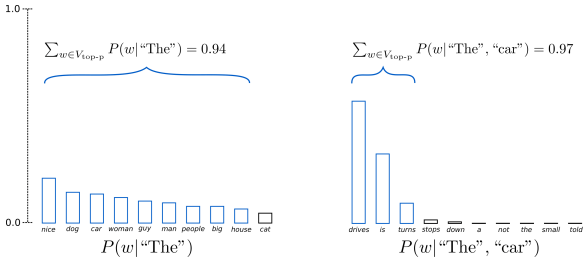
$$p(w|w_{1:t-1}) = \begin{cases} \frac{p(w|w_{1:t-1})}{\sum_{w' \in \text{top-K}} p(w'|w_{1:t-1})} & \text{si } w \in \text{top-K} \\ 0 & \text{sinon} \end{cases}$$

3. Échantillonner à partir de cette distribution tronquée

Top-K Sampling : Avantages et inconvénients

- **Avantages** :
  - Évite les tokens improbables
  - Équilibre entre diversité et qualité
- **Inconvénients** :
  - Le choix de K est arbitraire
  - Inadapté si la distribution est plate ou très pointue

Top-P Sampling : Adapter dynamiquement le nombre de choix



1. Sélectionner le **plus petit ensemble de tokens** dont la probabilité cumulée dépasse **p** (nucleus) :

$$V^{(p)} = \min_{V' \subset V} \left\{ V' \,\middle|\, \sum_{w \in V'} p(w|w_{1:t-1}) \geq p \right\}$$

2. Renormaliser les probabilités sur cet ensemble
3. Échantillonner parmi ces tokens

Top-P Sampling : Avantages et inconvénients

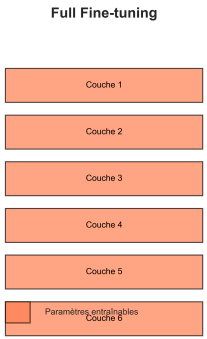
- **Avantages** :
  - S'adapte à la **forme de la distribution**
  - Plus de choix pour les distributions plates
  - Moins de choix pour les distributions pointues
  - Meilleur équilibre qualité/diversité
- **Inconvénients** :
  - Peu prévisible (nombre de tokens variable)

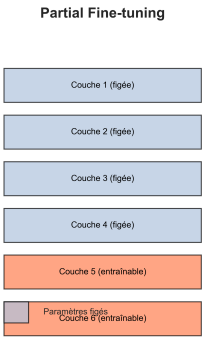
Pré-entraînement vs Fine-tuning : Deux phases distinctes

- **Pré-entraînement** :
  - Entraînement sur des **données massives et générales**
  - Objectif : apprendre des **représentations générales** du langage
  - Tâche auto-supervisée : prédiction du token suivant
  - Nécessite d'énormes ressources de calcul
  - Réalisé une seule fois par les grands laboratoires

- **Fine-tuning** :
  - Adaptation à une **tâche spécifique**
  - Données **étiquetées** pour la tâche cible
  - Préserve les connaissances générales
  - Ressources de calcul plus modestes
  - Meilleure performance sur la tâche cible

Full Fine-tuning vs Partial Fine-tuning





- **Full Fine-tuning** :
  - Mise à jour de **tous les paramètres** du modèle
  - Performances optimales
  - Nécessite beaucoup de **mémoire** et de **calcul**
  - Risque d'**oubli catastrophique**
- **Partial Fine-tuning** :
  - Seules certaines couches sont entraînées (souvent les dernières)
  - Moins de paramètres à mettre à jour
  - Économie de ressources
  - Préserve mieux les connaissances générales

Méthodes avancées de Fine-tuning : Efficacité et performances

- **Adapter** : Petits modules d'adaptation insérés entre les couches figées
- **LoRA** (Low-Rank Adaptation) : Matrices de rang faible en parallèle des poids principaux
- **Prompt Tuning** : Ajout de tokens apprenables au contexte d'entrée
- Avantages communs : économie de mémoire, transfert facilité, multitâche