VIP Cheatsheet : Transformeurs et Grands Modèles de Langage

Afshine Amidi et Shervine Amidi

26 mars 2025

Cette VIP cheatsheet donne un aperçu du contenu du livre intitulé « Super Study Guide : Transformeurs et Grands Modèles de Langage », qui contient environ 600 illustrations réparties sur 250 pages et explore en profondeur les concepts abordés ici.
Pour plus d'information, veuillez visiter : `https://superstudy.guide`.

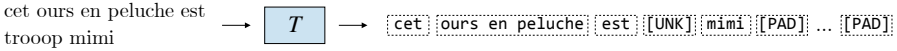
1 Fondations

1.1 Tokens

❒ **Définition** – Un *token* est une unité de texte indivisible, tel qu'un mot, un sous-mot ou un caractère, et fait partie d'un vocabulaire prédéfini.

Remarque : Le token inconnu `[UNK]` représente un morceau de texte qui ne fait pas partie du vocabulaire. De son côté, le token de remplissage `[PAD]` est utilisé pour combler les positions vides et permet de garantir des longueurs de séquences d'entrée cohérentes.

❒ **Tokeniseur** – Un *tokeniseur* T divise le texte en tokens selon un niveau de granularité arbitraire.



Voici les principaux types de tokeniseurs :

Type	Avantages	Inconvénients	Illustration
Mot	• Facile à interpréter • Séquence courte	• Vocabulaire de grande taille • Variation des mots n'est pas prise en compte	ours en peluche
Sous-mot	• Exploite les racines des mots • Représentations intuitives	• Séquence plus longue • Tokenisation plus complexe	ours en pe ##luche
Caractère Octet	• Pas de problème de token non-reconnu • Vocabulaire de petite taille	• Séquence beaucoup plus longue • Représentation vectorielle difficile à interpréter	o u r s _ e n _ p e l u c h e

Remarque : Byte-Pair Encoding (BPE) et Unigram sont des tokeniseurs couramment utilisés à l'échelle du sous-mot.

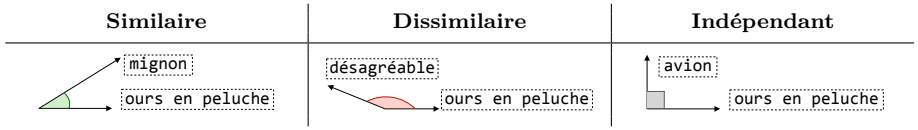
1.2 Embeddings

❒ **Définition** – Un *embedding* est une représentation vectorielle d'un élément, par exemple, un token ou une phrase. Elle est caractérisée par un vecteur $x \in \mathbb{R}^n$.

❒ **Similarité** – La *similarité cosinus* entre deux tokens t_1, t_2 est quantifiée par :

$$\text{similarité}(t_1, t_2) = \frac{t_1 \cdot t_2}{||t_1||\ ||t_2||} = \cos(\theta) \in [-1, 1]$$

L'angle θ caractérise la similarité entre les deux tokens :

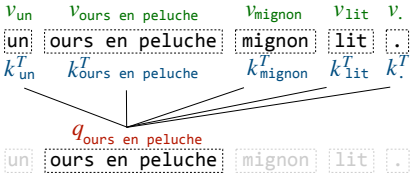


Remarque : Approximate Nearest Neighbors (ANN) et Locality Sensitive Hashing (LSH) sont des méthodes permettant d'approximer l'opération de similarité de manière efficace sur de grandes bases de données.

2 Transformeurs

2.1 Attention

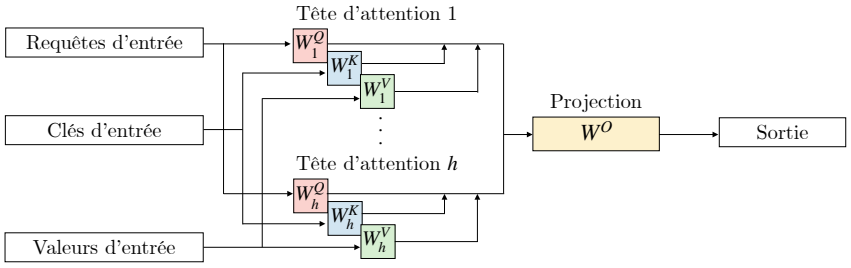
❒ **Formule** – Étant donné une *requête* q, on cherche à déterminer à quelle *clé* k la requête doit prêter "attention", en fonction de la *valeur* v qui lui est associée.



L'attention peut être efficacement calculée à l'aide des matrices Q, K, V, qui contiennent respectivement les requêtes q, les clés k et les valeurs v, ainsi que la dimension d_k des clés :

$$\text{attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

❒ **MHA** – Une *couche d'attention multi-têtes* (***M**ulti-**H**ead **A**ttention*, MHA) effectue des calculs d'attention à travers plusieurs têtes, puis projette le résultat dans l'espace de sortie.

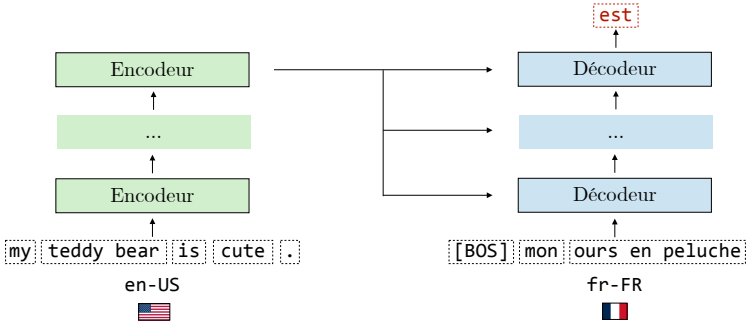


Elle est composée de h têtes d'attention ainsi que des matrices W^Q, W^K, W^V qui projettent l'entrée afin d'obtenir les requêtes Q, les clés K et les valeurs V. La projection finale est effectuée à l'aide de la matrice W^O.

Remarque : L'attention à requêtes groupées (Grouped-Query Attention, GQA) et l'attention à plusieurs requêtes (Multi-Query Attention, MQA) sont des variantes de l'attention multi-têtes (MHA) qui réduisent la charge de calcul en partageant les clés et les valeurs entre les têtes d'attention.

2.2 Architecture

❒ **Aperçu** – Le *Transformeur* est un modèle phare reposant sur le mécanisme d'auto-attention et est composé d'encodeurs et de décodeurs. L'encodeur calcule une représentation pertinente de l'entrée, qui est ensuite utilisée par le décodeur pour prédire le prochain token dans la séquence.



Remarque : Bien que le Transformeur ait été initialement proposé comme modèle pour les tâches de traduction, il est désormais utilisé dans de nombreuses autres applications.

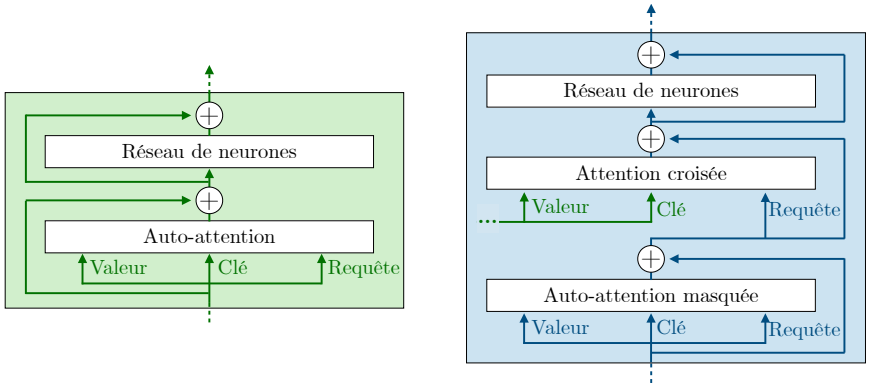
❒ **Composants** – L'*encodeur* et le *décodeur* sont deux composants fondamentaux du Transformeur et remplissent des rôles différents :

Encodeur

Les embeddings encodés représentent le sens de l'entrée

Décodeur

Les embeddings décodés représentent le sens de l'entrée ainsi que de la sortie prédite jusqu'à présent

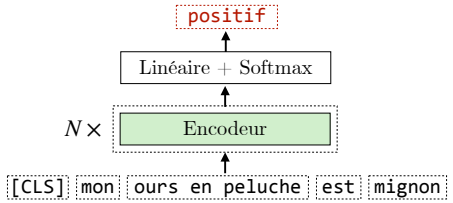


❒ **Embeddings positionnels** – Les *embeddings positionnels* indiquent la position du token dans la séquence et ont la même dimension que les embeddings des tokens. Ils peuvent être soit définis de manière arbitraire, soit appris à partir de données.

Remarque : L'embedding positionnel rotatif (Rotary Position Embedding, RoPE) est une variante populaire et efficace qui fait pivoter les vecteurs de requête et de clé pour incorporer des informations de position relative.

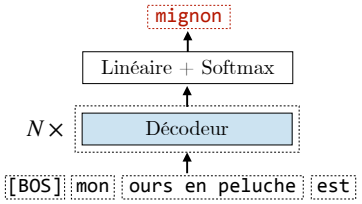
2.3 Variants

❒ **Modèles encodeurs** – BERT (***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*) est un modèle basé sur le Transformeur. Il est composé d'une pile d'encodeurs qui prend un texte en entrée et produit des embeddings riches en information qui peuvent ensuite être utilisés pour des tâches de classification.



Un token `[CLS]` est ajouté au début de la séquence pour capturer le sens de la phrase. Son embedding encodé est souvent utilisé dans des tâches en aval, comme l'analyse de sentiments.

❒ **Modèles décodeurs** – GPT (***G**enerative **P**re-trained **T**ransformer*) est un modèle autoregressif basé sur le Transformeur, composé d'une pile de décodeurs. Contrairement à BERT et ses dérivés, GPT traite tous les problèmes avec une formulation texte-à-texte.



La majorité des LLMs actuels les plus performants reposent sur une architecture exclusivement composée de décodeurs tel que LLaMA, Mistral, Gemma, DeepSeek, etc.

Remarque : Les modèles encodeur-décodeur, comme T5, sont également autoregressifs et partagent de nombreuses caractéristiques avec les modèles décodeurs.

2.4 Optimisations

❒ **Approximation de l'attention** – Les calculs d'attention sont en $\mathcal{O}(n^2)$, ce qui peut devenir coûteux lorsque la longueur n de la séquence est grande. Il existe deux méthodes principales pour approximer ces calculs :

— *Sparsité* : L'auto-attention ne s'effectue pas sur toute la séquence, mais uniquement entre les tokens les plus pertinents.

— *Bas rang* : La formule d'attention est simplifiée en un produit de matrices de rang faible, ce qui réduit la charge de calcul.

❒ **Flash attention** – *Flash attention* est une méthode exacte qui optimise les calculs d'attention en tirant parti du GPU, en utilisant la mémoire SRAM (***S**tatic **R**andom-**A**ccess **M**emory*), plus rapide, pour les opérations matricielles, avant d'écrire les résultats dans la mémoire HBM (***H**igh **B**andwidth **M**emory*), plus lente.

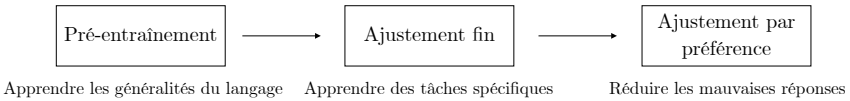
Remarque : En pratique, cela permet de réduire l'utilisation de la mémoire et d'accélérer les calculs.

3 Grands modèles de langage

3.1 Aperçu

❒ **Définition** – Un *grand modèle de langage* (***L**arge **L**anguage **M**odel*, LLM) est un modèle basé sur le Transformeur doté de fortes capacités en traitement du langage naturel. Il est qualifié de « grand » car il contient généralement des milliards de paramètres.

❒ **Cycle de vie** – Un LLM est entraîné en 3 étapes : *pré-entraînement* (*pretraining*), *ajustement fin* (*finetuning*) et *ajustement par préférence* (*preference tuning*).

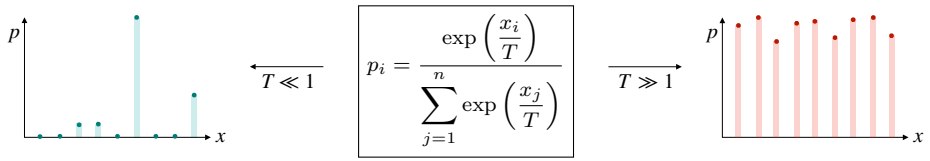


L'ajustement fin et l'ajustement par préférence sont des méthodes post-entraînement visant à *aligner le modèle* pour qu'il effectue des tâches précises.

3.2 Prompting

❒ **Longueur du contexte** – La *longueur du contexte* d'un modèle correspond au nombre maximal de tokens pouvant être inclus en l'entrée. Elle varie généralement de plusieurs dizaines de milliers à plusieurs millions de tokens.

❒ **Échantillonnage lors du décodage** – Les prédictions de tokens sont échantillonnées à partir de la distribution de probabilité prédite p_i, qui est contrôlée par l'hyperparamètre température T.



Remarque : Une température élevée produit des sorties plus créatives, tandis qu'une température faible produit des sorties plus déterministes.

❒ **Chaînage de pensée** – Le *chaînage de pensée* (***C**hain-**o**f-**T**hought*, CoT) est un processus de raisonnement dans lequel le modèle décompose un problème complexe en une série d'étapes intermédiaires. Cela aide le modèle à générer une réponse finale correcte. ***T**ree **o**f **T**houghts* (ToT) est une version plus avancée du CoT.

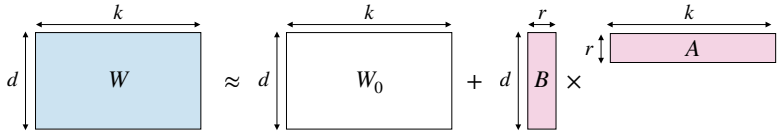
Remarque : L'auto-cohérence (self-consistency) est une méthode qui agrège les réponses issues de différents chemins de raisonnement CoT.

3.3 Ajustement fin

❒ **SFT** – L'*ajustement fin supervisé* (***S**upervised **F**ine**T**uning*, SFT) est une méthode post-entraînement qui aligne le comportement du modèle avec une tâche finale. Il se repose sur des paires de données d'entrée-sortie de haute qualité, qui sont alignées avec la tâche en question.

Remarque : Lorsque les données SFT sont composées d'instructions, cette étape est appelée « ajustement par instructions » (instruction tuning).

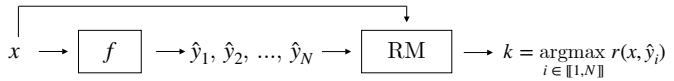
❒ **PEFT** – L'*ajustement fin efficace en paramètres* (***P**arameter-**E**fficient **F**ine**T**uning*, PEFT) est une catégorie de méthodes permettant d'exécuter le SFT de manière efficace. En particulier, l'*adaptation à faible rang* (***L**ow-**R**ank **A**daptation*, LoRA) approxime les coefficients de W en fixant W_0 et en apprenant des matrices de bas rang A et B :



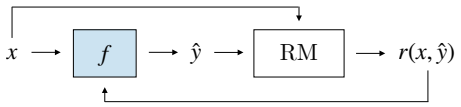
Remarque : On compte l'ajustement de préfixe (prefix tuning) et l'insertion de couches d'adaptateurs (adapter layer insertion) parmi les autres techniques PEFT.

3.4 Preference tuning

❒ **Modèle de récompense** – Un *modèle de récompense* (***R**eward **M**odel*, RM) est un modèle qui prédit la mesure dans laquelle une sortie $\hat{y}$ est alignée avec le comportement souhaité, étant donné une entrée x. L'échantillonnage ***B**est-**o**f-**N*** (BoN), aussi appelé *échantillonnage par rejet* (*rejection sampling*), est une méthode qui utilise un modèle de récompense pour sélectionner la meilleure réponse parmi N générations.



❒ **Apprentissage par renforcement** – L'*apprentissage par renforcement* (***R**einforcement **L**earning*, RL) est une approche qui utilise un RM et met à jour le modèle f en fonction des récompenses associées à ses sorties générées. Si le RM repose sur des préférences humaines, ce processus est appelé *apprentissage par renforcement à partir de rétroaction humaine* (***R**einforcement **L**earning from **H**uman **F**eedback*, RLHF).

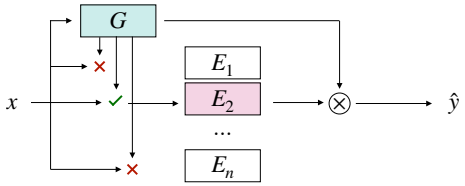


L'*optimisation de politique proximale* (***P**roximal **P**olicy **O**ptimization*, PPO) est un algorithme de RL populaire qui encourage les récompenses plus élevées tout en maintenant le modèle proche du modèle de base, ce qui permet d'éviter le « reward hacking ».

Remarque : Il existe également des approches supervisées, comme l'optimisation de préférence directe (Direct Preference Optimization, DPO), qui combinent le RM et le RL en une seule étape supervisée.

3.5 Optimisations

❒ **Mélange d'experts** – Un *mélange d'experts* (***M**ixture **o**f **E**xperts*, MoE) est un modèle qui active seulement une partie de ses neurones lors de l'étape d'inférence. Il se repose sur une porte G (gate) et des experts $E_1, ..., E_n$.



$$\hat{y} = \sum_{i=1}^{n} G(x)_i E_i(x)$$

Les LLMs basés sur des MoEs utilisent ce mécanisme de porte dans leurs réseaux de neurones à propagation directe (***F**eed-**F**orward **N**eural **N**etwork*, FFNNs).

Remarque : Il est connu que l'entraînement d'un LLM basé sur MoE est difficile, comme peut en témoigner l'article de LLaMA dont les auteurs expliquent ne pas utiliser cette architecture malgré son efficacité lors de l'inférence.

❒ **Distillation** – La *distillation* est un processus dans lequel un petit modèle élève S est entraîné à partir des sorties de prédiction d'un grand modèle enseignant T. Il est entraîné en utilisant la divergence de Kullback-Leibler (KL) :

$$\mathrm{KL}(\hat{y}_T || \hat{y}_S) = \sum_i \hat{y}_T^{(i)} \log\left(\frac{\hat{y}_T^{(i)}}{\hat{y}_S^{(i)}}\right)$$

Remarque : Les labels d'entraînement sont considérés comme étant des labels « doux » (soft labels), car ils représentent des probabilités de classe.

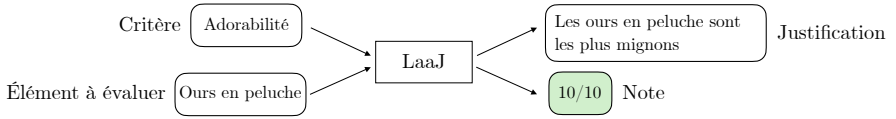
❒ **Quantification** – La *quantification de modèle* (*model quantization*) est une catégorie de techniques visant à réduire la précision des coefficients du modèle tout en limitant l'impact sur sa performance. Cela permet de réduire l'empreinte mémoire du modèle et d'accélérer son inférence.

Remarque : QLoRA est une variante quantifiée de LoRA et est couramment utilisée.

4 Applications

4.1 LLM comme Juge

❒ **Définition** – *LLM comme Juge* (***LLM-a**s-**a**-**J**udge*, LaaJ) est une méthode qui utilise un LLM pour noter des sorties données selon certains critères fournis. De manière notable, il est également capable de générer une justification pour sa note, ce qui améliore son interprétabilité.



Contrairement aux métriques de l'ère pré-LLM telles que ***R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation* (ROUGE), LaaJ ne nécessite aucun texte de référence, ce qui le rend pratique pour évaluer n'importe quel type de tâche. En particulier, LaaJ montre une forte corrélation avec les évaluations humaines lorsqu'il s'appuie sur un grand modèle puissant (par exemple GPT-4), car il utilise ses capacités de raisonnement pour obtenir une bonne performance.

Remarque : Même si LaaJ est utile pour effectuer des séries rapides d'évaluations, il est toutefois important de surveiller l'alignement entre les sorties de LaaJ et les évaluations humaines pour éviter toute divergence.

❒ **Biais courants** – Les modèles LaaJ peuvent présenter les biais suivants :

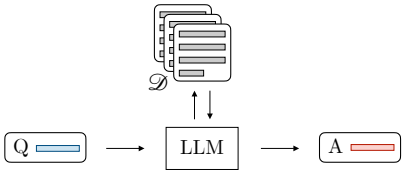
	Biais de position	**Biais de verbosité**	**Biais de préférence de soi**
Problème	Favorise la première position dans les comparaisons par paires	Favorise les contenus plus longs	Favorise les sorties générées par lui-même
Solution	Moyenne de la métrique sur des positions aléatoires	Ajouter une pénalité sur la longueur de la sortie	Utiliser un juge issu d'un autre modèle de base

Un remède à ces problèmes peut consister à personnaliser un LaaJ par un ajustement fin, mais cela demande beaucoup d'efforts.

Remarque : La liste des biais ci-dessus n'est pas exhaustive.

4.2 RAG

❒ **Définition** – La *génération augmentée par récupération* (***R**etrieval-**A**ugmented **G**eneration*, RAG) est une méthode qui permet à un LLM d'accéder à des connaissances externes pertinentes pour répondre à une question donnée. Cela est particulièrement utile lorsque l'on souhaite intégrer des informations postérieures à la date de coupure des connaissances pré-entraînées du LLM.



Étant donné une base de connaissances $\mathscr{D}$ et une question, un module de **R**écupération extrait les documents les plus pertinents, puis **A**ugmente l'invite avec ces informations avant de **G**énérer la réponse.

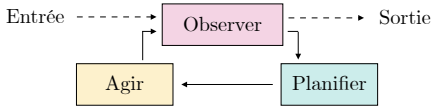
Remarque : La phase de récupération repose généralement sur des embeddings issus de modèles encodeurs.

❒ **Hyperparamètres** – La base de connaissances $\mathscr{D}$ est initialisée en découpant les documents en segments (*chunks*) de taille n_c et en les encodant sous forme de vecteurs de dimension $\mathbb{R}^d$.

4.3 Agents

❒ **Définition** – Un *agent* est un système qui poursuit des objectifs et accomplit des tâches de manière autonome pour le compte d'un utilisateur. Il peut utiliser différentes chaînes d'appels à des LLMs pour le faire.

❒ **ReAct** – ***Reason*** + ***Act*** (ReAct) est une méthode qui permet d'enchaîner plusieurs appels à un LLM pour accomplir des tâches complexes :



Cette méthode se compose des étapes suivantes :

- *Observer* : Synthétiser les actions précédentes et énoncer explicitement ce qui est actuellement connu.
- *Planifier* : Détailler les tâches à accomplir et les outils à utiliser.
- *Agir* : Réaliser une action via une API ou rechercher des informations pertinentes dans une base de connaissances.

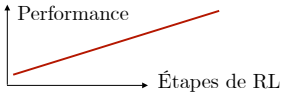
Remarque : L'évaluation d'un système agentique est difficile. Toutefois, elle peut être effectuée à la fois au niveau des composants via les entrées-sorties locales et au niveau du système via les chaînes d'appels.

4.4 Modèles de raisonnement

❒ **Définition** – Un *modèle de raisonnement* est un modèle qui s'appuie sur des traces de raisonnement de type CoT pour résoudre des tâches plus complexes en mathématiques, en programmation ou en logique. Des exemples de modèles de raisonnement incluent la série o d'OpenAI, DeepSeek-R1 et Gemini Flash Thinking de Google.

Remarque : En plus de sa réponse finale, DeepSeek-R1 retourne également le détail de son raisonnement entre les balises `<think>`.

❒ **Extension** – Deux types de méthodes d'extension sont utilisées pour améliorer les capacités de raisonnement :

	Description	**Illustration**
Extension à l'entraînement	Faire de l'apprentissage par renforcement plus longtemps pour que le modèle apprenne à produire des traces CoT avant de donner une réponse	
Extension à l'inférence	Laisser le modèle « réfléchir » plus longtemps avant de donner une réponse, en utilisant des mots-clés comme « `Attends` »