RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale
Togo Data Lab

Masterclass organisée par :
Togo Data Lab
Fondements Mathématiques des Transformers et des LLMs

UNIVERSITE DE KARA

Module 3 : Des Transformers aux LLMs : fondements et pré-entraînement

Présentée par : Tiebekabe Pagdame
Enseignant-chercheur - Université de Kara

Dates : 14-15 juillet 2025

RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale
Togo Data Lab

RÉPUBLIQUE TOGOLAISE
Ministère de l'Économie Numérique et de la Transformation Digitale

atd AGENCE TOGO DIGITAL

Sommaire

Modèle de Langage Probabiliste

- Un **modèle de langage** est une distribution de probabilité définie sur les séquences de mots $(w_1, w_2, \ldots, w_T)$, avec w_t appartenant à un vocabulaire fini $\mathcal{V}$.
- Objectif : apprendre la distribution conjointe
$$P(w_1, w_2, \ldots, w_T)$$
à partir d'un corpus de textes, afin de modéliser les régularités syntaxiques, sémantiques ou pragmatiques de la langue.

Formulation mathématique par la règle de chaîne (chaîne de Markov de longueur variable)

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

- Hypothèse de Markov d'ordre n :
$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$
Cela simplifie l'apprentissage et réduit le nombre de paramètres, mais limite la capacité de modélisation à des dépendances de longueur n.

Modèle de Langage Probabiliste

- Un **modèle de langage** est une distribution de probabilité définie sur les séquences de mots $(w_1, w_2, \ldots, w_T)$, avec w_t appartenant à un vocabulaire fini $\mathcal{V}$.
- Objectif : apprendre la distribution conjointe

$$P(w_1, w_2, \ldots, w_T)$$

à partir d'un corpus de textes, afin de modéliser les régularités syntaxiques, sémantiques ou pragmatiques de la langue.

Formulation mathématique par la règle de chaîne (chaîne de Markov de longueur variable)

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

- Hypothèse de Markov d'ordre n :

$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$

Cela simplifie l'apprentissage et réduit le nombre de paramètres, mais limite la capacité de modélisation à des dépendances de longueur n.

Modèle de Langage Probabiliste

- Un **modèle de langage** est une distribution de probabilité définie sur les séquences de mots $(w_1, w_2, \ldots, w_T)$, avec w_t appartenant à un vocabulaire fini $\mathcal{V}$.
- Objectif : apprendre la distribution conjointe
$$P(w_1, w_2, \ldots, w_T)$$
à partir d'un corpus de textes, afin de modéliser les régularités syntaxiques, sémantiques ou pragmatiques de la langue.

Formulation mathématique par la règle de chaîne (chaîne de Markov de longueur variable)

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \ldots, w_{t-1})$$

- Hypothèse de Markov d'ordre n :
$$P(w_t \mid w_1, \ldots, w_{t-1}) \approx P(w_t \mid w_{t-n+1}, \ldots, w_{t-1})$$
Cela simplifie l'apprentissage et réduit le nombre de paramètres, mais limite la capacité de modélisation à des dépendances de longueur n.

Exemple simple de modèle de langage

- **Vocabulaire** : $\mathcal{V} = \{\texttt{je}, \texttt{suis}, \texttt{content}\}$
- **Corpus** d'entraînement :
  - "je suis content"
  - "je suis"
- On extrait les bigrammes suivants :

$$(\texttt{je}, \texttt{suis}) \text{ (x2)}, \quad (\texttt{suis}, \texttt{content}) \text{ (x1)}$$

- D'où :

$$\text{count}(\texttt{je suis}) = 2, \quad \text{count}(\texttt{suis content}) = 1$$

$$\text{count}(\texttt{je}) = 2, \quad \text{count}(\texttt{suis}) = 2$$

- Estimation des probabilités (bigrammes) :

$$P(\texttt{suis} \mid \texttt{je}) = \frac{2}{2} = 1, \quad P(\texttt{content} \mid \texttt{suis}) = \frac{1}{2}$$

Les modèles neuronaux généralisent ce mécanisme en apprenant les probabilités via des paramètres optimisés.

Exemple simple de modèle de langage

- **Vocabulaire** : $\mathcal{V} = \{\texttt{je}, \texttt{suis}, \texttt{content}\}$
- **Corpus** d'entraînement :
  - "je suis content"
  - "je suis"
- On extrait les bigrammes suivants :

$$(\texttt{je}, \texttt{suis}) \text{ (x2)}, \quad (\texttt{suis}, \texttt{content}) \text{ (x1)}$$

- D'où :

$$\text{count}(\texttt{je suis}) = 2, \quad \text{count}(\texttt{suis content}) = 1$$

$$\text{count}(\texttt{je}) = 2, \quad \text{count}(\texttt{suis}) = 2$$

- Estimation des probabilités (bigrammes) :

$$P(\texttt{suis} \mid \texttt{je}) = \frac{2}{2} = 1, \quad P(\texttt{content} \mid \texttt{suis}) = \frac{1}{2}$$

Les modèles neuronaux généralisent ce mécanisme en apprenant les probabilités via des paramètres optimisés.

Exemple simple de modèle de langage

- **Vocabulaire** : $\mathcal{V} = \{\texttt{je}, \texttt{suis}, \texttt{content}\}$
- **Corpus** d'entraînement :
  - "je suis content"
  - "je suis"
- On extrait les bigrammes suivants :

$$(\texttt{je}, \texttt{suis}) \text{ (x2)}, \quad (\texttt{suis}, \texttt{content}) \text{ (x1)}$$

- D'où :

$$\text{count}(\texttt{je suis}) = 2, \quad \text{count}(\texttt{suis content}) = 1$$

$$\text{count}(\texttt{je}) = 2, \quad \text{count}(\texttt{suis}) = 2$$

- Estimation des probabilités (bigrammes) :

$$P(\texttt{suis} \mid \texttt{je}) = \frac{2}{2} = 1, \quad P(\texttt{content} \mid \texttt{suis}) = \frac{1}{2}$$

Les modèles neuronaux généralisent ce mécanisme en apprenant les probabilités via des paramètres optimisés.

Exemple simple de modèle de langage

- **Vocabulaire** : $\mathcal{V} = \{\texttt{je}, \texttt{suis}, \texttt{content}\}$
- **Corpus** d'entraînement :
  - "je suis content"
  - "je suis"
- On extrait les bigrammes suivants :

$$(\texttt{je}, \texttt{suis}) \text{ (x2)}, \quad (\texttt{suis}, \texttt{content}) \text{ (x1)}$$

- D'où :

$$\text{count}(\texttt{je suis}) = 2, \quad \text{count}(\texttt{suis content}) = 1$$

$$\text{count}(\texttt{je}) = 2, \quad \text{count}(\texttt{suis}) = 2$$

- Estimation des probabilités (bigrammes) :

$$P(\texttt{suis} \mid \texttt{je}) = \frac{2}{2} = 1, \quad P(\texttt{content} \mid \texttt{suis}) = \frac{1}{2}$$

Les modèles neuronaux généralisent ce mécanisme en apprenant les probabilités via des paramètres optimisés.

Fonction de Vraisemblance et Entraînement

Objectif : Maximum de vraisemblance

Étant donné une séquence $(w_1, w_2, \ldots, w_T)$, on maximise la vraisemblance du modèle P_θ :

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

où θ représente les paramètres du modèle (poids du réseau, biais, etc.).

Log-vraisemblance (plus stable numériquement)

$$\log \mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

- En pratique, on minimise la **négative log-vraisemblance** (fonction de perte) :

$$\mathcal{J}(\theta) = -\sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

Cette fonction est utilisée dans les algorithmes d'apprentissage (descente de gradient stochastique).

Fonction de Vraisemblance et Entraînement

Objectif : Maximum de vraisemblance

Étant donné une séquence $(w_1, w_2, \ldots, w_T)$, on maximise la vraisemblance du modèle P_θ :

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

où θ représente les paramètres du modèle (poids du réseau, biais, etc.).

Log-vraisemblance (plus stable numériquement)

$$\log \mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

- En pratique, on minimise la **négative log-vraisemblance** (fonction de perte) :

$$\mathcal{J}(\theta) = -\sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

Cette fonction est utilisée dans les algorithmes d'apprentissage (descente de gradient stochastique).

Fonction de Vraisemblance et Entraînement

Objectif : Maximum de vraisemblance

Étant donné une séquence $(w_1, w_2, \ldots, w_T)$, on maximise la vraisemblance du modèle P_θ :

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

où θ représente les paramètres du modèle (poids du réseau, biais, etc.).

Log-vraisemblance (plus stable numériquement)

$$\log \mathcal{L}(\theta) = \sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

- En pratique, on minimise la **négative log-vraisemblance** (fonction de perte) :

$$\mathcal{J}(\theta) = -\sum_{t=1}^{T} \log P_\theta(w_t \mid w_1, \ldots, w_{t-1})$$

Cette fonction est utilisée dans les algorithmes d'apprentissage (descente de gradient stochastique).

Résumé visuel

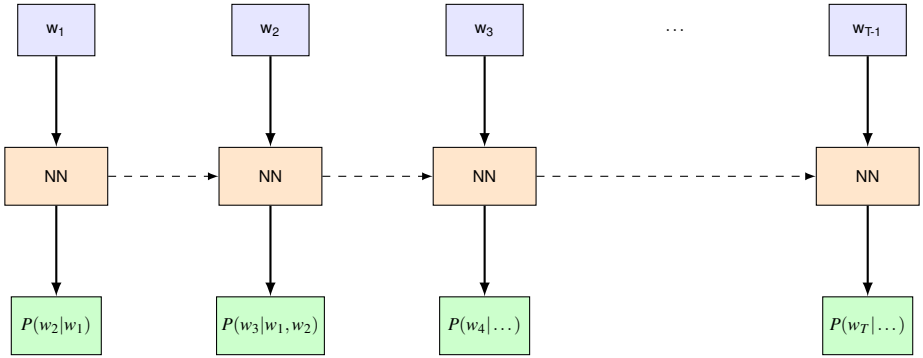


Figure – Architecture d'un modèle de langage probabiliste basé sur réseau neuronal.

- Les probabilités sont estimées par un réseau neuronal (RNN, Transformer, etc.)
- L'entraînement se fait par maximisation de la vraisemblance

Problème de la Tokenisation

- Dans les LLMs, le texte est converti en unités élémentaires : les **tokens**.
- Un token peut être un mot, une syllabe, un caractère ou une sous-unité morphologique.
- Objectifs :
  - Réduire la taille du vocabulaire.
  - Gérer les mots inconnus ou rares (*Out-of-Vocabulary*).
  - Maximiser la réutilisabilité statistique des morceaux.

Problème : Comment découper un texte efficacement tout en gardant une expressivité linguistique et une efficacité computationnelle ?

Problème de la Tokenisation

- Dans les LLMs, le texte est converti en unités élémentaires : les **tokens**.
- Un token peut être un mot, une syllabe, un caractère ou une sous-unité morphologique.
- Objectifs :
  - Réduire la taille du vocabulaire.
  - Gérer les mots inconnus ou rares (*Out-of-Vocabulary*).
  - Maximiser la réutilisabilité statistique des morceaux.

Problème : Comment découper un texte efficacement tout en gardant une expressivité linguistique et une efficacité computationnelle ?

Tokenisation par mots vs. caractères vs. sous-mots

Tokenisation par mots

- Facile à interpréter.
- Taille de vocabulaire énorme.
- Problèmes avec les mots rares ou inconnus.

Tokenisation par caractères

- Très petit vocabulaire.
- Longues séquences.
- Perte de structure linguistique.

Tokenisation par sous-mots (subwords)

- Compromis entre expressivité et efficacité.
- Basée sur la fréquence des co-occurrences.
- Utilisée par les LLMs modernes : GPT, BERT, T5.

Les sous-mots permettent d'équilibrer la couverture lexicale et la généralisation.

Tokenisation par mots vs. caractères vs. sous-mots

Tokenisation par mots

- Facile à interpréter.
- Taille de vocabulaire énorme.
- Problèmes avec les mots rares ou inconnus.

Tokenisation par caractères

- Très petit vocabulaire.
- Longues séquences.
- Perte de structure linguistique.

Tokenisation par sous-mots (subwords)

- Compromis entre expressivité et efficacité.
- Basée sur la fréquence des co-occurrences.
- Utilisée par les LLMs modernes : GPT, BERT, T5.

Les sous-mots permettent d'équilibrer la couverture lexicale et la généralisation.

Byte Pair Encoding (BPE)

- Méthode initialement conçue pour la compression de texte (Gage, 1994), adaptée au traitement automatique des langues par Sennrich et al. (2016).
- But : apprendre un vocabulaire de **sous-mots** (subwords) à partir d'un corpus en découpant ou fusionnant les mots selon des motifs statistiques.
- **Idée clé** : fusionner les paires de symboles les plus fréquentes dans le corpus pour construire des unités plus longues.
- **Algorithme BPE formalisé :**
  1. Initialiser un vocabulaire de symboles $\mathcal{V}_0$ (souvent les caractères individuels).
  2. À chaque itération t, repérer la paire de symboles adjacents (a, b) la plus fréquente dans le corpus $C^{(t)}$:

$$(a_t, b_t) = \arg\max_{(a,b)} \text{count}_{C^{(t)}}(a, b)$$

  3. Ajouter le nouveau symbole ab à $\mathcal{V}_t$, et remplacer toutes les occurrences de $a\ b$ par ab dans $C^{(t)}$ pour obtenir $C^{(t+1)}$.
  4. Répéter jusqu'à atteindre un vocabulaire de taille $|\mathcal{V}|$ prédéfini.
- **Exemple (toy)** :

  `l o w  l o w e r  n e w e s t  w i d e s t`
  - Étape 1 : compter les paires fréquentes $\Rightarrow$ `(l,o)`, `(o,w)`, `(e,s)`, etc.
  - Étape 2 : fusionner la paire la plus fréquente, disons `l o` $\rightarrow$ `lo`
  - Corpus devient : `lo w  lo w e r  n e w e s t  w i d e s t`

Remarque : BPE est rapide, déterministe et non probabiliste, mais peut générer des unités fréquentes pertinentes même pour des mots inconnus (OOV).

Byte Pair Encoding (BPE)

- Méthode initialement conçue pour la compression de texte (Gage, 1994), adaptée au traitement automatique des langues par Sennrich et al. (2016).
- But : apprendre un vocabulaire de **sous-mots** (subwords) à partir d'un corpus en découpant ou fusionnant les mots selon des motifs statistiques.
- **Idée clé** : fusionner les paires de symboles les plus fréquentes dans le corpus pour construire des unités plus longues.
- **Algorithme BPE formalisé :**
  1. Initialiser un vocabulaire de symboles $\mathcal{V}_0$ (souvent les caractères individuels).
  2. À chaque itération t, repérer la paire de symboles adjacents (a, b) la plus fréquente dans le corpus $C^{(t)}$:
  $$(a_t, b_t) = \arg\max_{(a,b)} \text{count}_{C^{(t)}}(a, b)$$
  3. Ajouter le nouveau symbole ab à $\mathcal{V}_t$, et remplacer toutes les occurrences de $a\ b$ par ab dans $C^{(t)}$ pour obtenir $C^{(t+1)}$.
  4. Répéter jusqu'à atteindre un vocabulaire de taille $|\mathcal{V}|$ prédéfini.
- **Exemple (toy)** :
  `l o w  l o w e r  n e w e s t  w i d e s t`
  - Étape 1 : compter les paires fréquentes ⇒ `(l,o)`, `(o,w)`, `(e,s)`, etc.
  - Étape 2 : fusionner la paire la plus fréquente, disons `l o` → `lo`
  - Corpus devient : `lo w  lo w e r  n e w e s t  w i d e s t`

Remarque : BPE est rapide, déterministe et non probabiliste, mais peut générer des unités fréquentes pertinentes même pour des mots inconnus (OOV).

Byte Pair Encoding (BPE)

- Méthode initialement conçue pour la compression de texte (Gage, 1994), adaptée au traitement automatique des langues par Sennrich et al. (2016).
- But : apprendre un vocabulaire de **sous-mots** (subwords) à partir d'un corpus en découpant ou fusionnant les mots selon des motifs statistiques.
- **Idée clé** : fusionner les paires de symboles les plus fréquentes dans le corpus pour construire des unités plus longues.
- **Algorithme BPE formalisé :**
  1. Initialiser un vocabulaire de symboles $\mathcal{V}_0$ (souvent les caractères individuels).
  2. À chaque itération t, repérer la paire de symboles adjacents (a, b) la plus fréquente dans le corpus $C^{(t)}$:
  $$(a_t, b_t) = \arg\max_{(a,b)} \text{count}_{C^{(t)}}(a, b)$$
  3. Ajouter le nouveau symbole ab à $\mathcal{V}_t$, et remplacer toutes les occurrences de $a\ b$ par ab dans $C^{(t)}$ pour obtenir $C^{(t+1)}$.
  4. Répéter jusqu'à atteindre un vocabulaire de taille $|\mathcal{V}|$ prédéfini.
- **Exemple (toy)** :
  `l o w  l o w e r  n e w e s t  w i d e s t`
  - Étape 1 : compter les paires fréquentes ⇒ `(l,o)`, `(o,w)`, `(e,s)`, etc.
  - Étape 2 : fusionner la paire la plus fréquente, disons `l o` → `lo`
  - Corpus devient : `lo w  lo w e r  n e w e s t  w i d e s t`

Remarque : BPE est rapide, déterministe et non probabiliste, mais peut générer des unités fréquentes pertinentes même pour des mots inconnus (OOV).

WordPiece : Amélioration probabiliste de BPE

- Méthode utilisée par Google dans les modèles tels que BERT, introduite dans le contexte de la reconnaissance vocale (Schuster & Nakajima, 2012).
- Objectif : construire un vocabulaire optimal de sous-unités qui maximise la probabilité du corpus.
- **Formulation probabiliste :**
  Soit un corpus $C = \{w^{(1)}, w^{(2)}, \dots\}$, et un vocabulaire courant $\mathcal{V}$ de sous-unités.
  Pour chaque mot w, on peut l'écrire comme une séquence de sous-unités :

$$w = t_1 t_2 \dots t_k, \quad \text{où } t_i \in \mathcal{V}$$

  On cherche à maximiser :

$$\mathcal{L}(\mathcal{V}) = \sum_{w \in C} \log P(t_1, \dots, t_k)$$

  Typiquement, on utilise une modélisation de type unigramme :

$$P(t_1, \dots, t_k) = \prod_{i=1}^{k} P(t_i)$$

- **Algorithme :**
  1. Initialiser $\mathcal{V}_0$ à l'ensemble des caractères.
  2. À chaque itération, ajouter le token (sous-mot) candidat t^* qui augmente le plus la log-vraisemblance du corpus :

$$t^* = \arg\max_{t \notin \mathcal{V}} \Delta\mathcal{L}(t)$$

  3. Répéter jusqu'à obtenir la taille de vocabulaire souhaitée.
- **Avantages par rapport à BPE** :
  - Prise en compte directe de la probabilité jointe des décompositions.
  - Plus robuste aux biais de fréquence brute.
  - Meilleure couverture du vocabulaire en entraînement / test.

WordPiece : Amélioration probabiliste de BPE

- Méthode utilisée par Google dans les modèles tels que BERT, introduite dans le contexte de la reconnaissance vocale (Schuster & Nakajima, 2012).
- Objectif : construire un vocabulaire optimal de sous-unités qui maximise la probabilité du corpus.
- **Formulation probabiliste :**
  Soit un corpus $C = \{w^{(1)}, w^{(2)}, \dots\}$, et un vocabulaire courant $\mathcal{V}$ de sous-unités.
  Pour chaque mot w, on peut l'écrire comme une séquence de sous-unités :

$$w = t_1 t_2 \dots t_k, \quad \text{où } t_i \in \mathcal{V}$$

  On cherche à maximiser :

$$\mathcal{L}(\mathcal{V}) = \sum_{w \in C} \log P(t_1, \dots, t_k)$$

  Typiquement, on utilise une modélisation de type unigramme :

$$P(t_1, \dots, t_k) = \prod_{i=1}^{k} P(t_i)$$

- **Algorithme :**
  1. Initialiser $\mathcal{V}_0$ à l'ensemble des caractères.
  2. À chaque itération, ajouter le token (sous-mot) candidat t^* qui augmente le plus la log-vraisemblance du corpus :

$$t^* = \arg\max_{t \notin \mathcal{V}} \Delta\mathcal{L}(t)$$

  3. Répéter jusqu'à obtenir la taille de vocabulaire souhaitée.
- **Avantages par rapport à BPE** :
  - Prise en compte directe de la probabilité jointe des décompositions.
  - Plus robuste aux biais de fréquence brute.
  - Meilleure couverture du vocabulaire en entraînement / test.

WordPiece : Amélioration probabiliste de BPE

- Méthode utilisée par Google dans les modèles tels que BERT, introduite dans le contexte de la reconnaissance vocale (Schuster & Nakajima, 2012).
- Objectif : construire un vocabulaire optimal de sous-unités qui maximise la probabilité du corpus.
- **Formulation probabiliste :**
  Soit un corpus $C = \{w^{(1)}, w^{(2)}, \dots\}$, et un vocabulaire courant $\mathcal{V}$ de sous-unités.
  Pour chaque mot w, on peut l'écrire comme une séquence de sous-unités :

$$w = t_1 t_2 \dots t_k, \quad \text{où } t_i \in \mathcal{V}$$

  On cherche à maximiser :

$$\mathcal{L}(\mathcal{V}) = \sum_{w \in C} \log P(t_1, \dots, t_k)$$

  Typiquement, on utilise une modélisation de type unigramme :

$$P(t_1, \dots, t_k) = \prod_{i=1}^{k} P(t_i)$$

- **Algorithme :**
  1. Initialiser $\mathcal{V}_0$ à l'ensemble des caractères.
  2. À chaque itération, ajouter le token (sous-mot) candidat t^* qui augmente le plus la log-vraisemblance du corpus :

$$t^* = \arg\max_{t \notin \mathcal{V}} \Delta\mathcal{L}(t)$$

  3. Répéter jusqu'à obtenir la taille de vocabulaire souhaitée.
- **Avantages par rapport à BPE** :
  - Prise en compte directe de la probabilité jointe des décompositions.
  - Plus robuste aux biais de fréquence brute.
  - Meilleure couverture du vocabulaire en entraînement / test.

WordPiece : Amélioration probabiliste de BPE

- Méthode utilisée par Google dans les modèles tels que BERT, introduite dans le contexte de la reconnaissance vocale (Schuster & Nakajima, 2012).
- Objectif : construire un vocabulaire optimal de sous-unités qui maximise la probabilité du corpus.
- **Formulation probabiliste :**
  Soit un corpus $C = \{w^{(1)}, w^{(2)}, \dots\}$, et un vocabulaire courant $\mathcal{V}$ de sous-unités.
  Pour chaque mot w, on peut l'écrire comme une séquence de sous-unités :

$$w = t_1 t_2 \dots t_k, \quad \text{où } t_i \in \mathcal{V}$$

  On cherche à maximiser :

$$\mathcal{L}(\mathcal{V}) = \sum_{w \in C} \log P(t_1, \dots, t_k)$$

  Typiquement, on utilise une modélisation de type unigramme :

$$P(t_1, \dots, t_k) = \prod_{i=1}^{k} P(t_i)$$

- **Algorithme :**
  1. Initialiser $\mathcal{V}_0$ à l'ensemble des caractères.
  2. À chaque itération, ajouter le token (sous-mot) candidat t^* qui augmente le plus la log-vraisemblance du corpus :

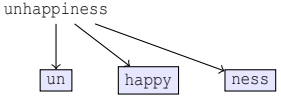
$$t^* = \arg\max_{t \notin \mathcal{V}} \Delta\mathcal{L}(t)$$

  3. Répéter jusqu'à obtenir la taille de vocabulaire souhaitée.
- **Avantages par rapport à BPE** :
  - Prise en compte directe de la probabilité jointe des décompositions.
  - Plus robuste aux biais de fréquence brute.
  - Meilleure couverture du vocabulaire en entraînement / test.

BPE vs WordPiece : comparaison

Critère	**BPE**	**WordPiece**
Principe	Fusion des paires fréquentes	Maximisation de la probabilité jointe
Approche	Déterministe, fréquence brute	Probabiliste (vraisemblance)
Vitesse	Très rapide	Plus lente
Utilisation	GPT, RoBERTa	BERT, ALBERT

Visualisation : Exemple de Tokenisation Subword



Découpe par BPE ou WordPiece

Impact sur les modèles LLM

- La tokenisation affecte :
  - La longueur des séquences d'entrée.
  - La couverture linguistique.
  - La capacité de généralisation du modèle.
- Choix du vocabulaire $\rightarrow$ compromis entre complexité et expressivité.
- Exemples :
  - GPT utilise un vocabulaire de $\sim$ 50k subwords (BPE).
  - BERT utilise WordPiece avec $\sim$ 30k tokens.

Conséquence : Le succès des LLMs dépend grandement de la stratégie de tokenisation.

Impact sur les modèles LLM

- La tokenisation affecte :
  - La longueur des séquences d'entrée.
  - La couverture linguistique.
  - La capacité de généralisation du modèle.
- Choix du vocabulaire $\rightarrow$ compromis entre complexité et expressivité.
- Exemples :
  - GPT utilise un vocabulaire de $\sim$ 50k subwords (BPE).
  - BERT utilise WordPiece avec $\sim$ 30k tokens.

Conséquence : Le succès des LLMs dépend grandement de la stratégie de tokenisation.

Apprentissage auto-supervisé

- L'auto-supervision consiste à créer automatiquement des labels à partir des données elles-mêmes.
- Très utile dans les modèles de langage : aucun besoin d'annotations humaines coûteuses.
- Deux paradigmes majeurs :
  - **Prédiction causale** : prédire le prochain token à partir du contexte passé.
  - **Prédiction masquée** : prédire les tokens manquants dans une séquence.

Ces stratégies d'entraînement sont à la base des grands modèles comme GPT et BERT.

Apprentissage auto-supervisé

- L'auto-supervision consiste à créer automatiquement des labels à partir des données elles-mêmes.
- Très utile dans les modèles de langage : aucun besoin d'annotations humaines coûteuses.
- Deux paradigmes majeurs :
  - **Prédiction causale** : prédire le prochain token à partir du contexte passé.
  - **Prédiction masquée** : prédire les tokens manquants dans une séquence.

Ces stratégies d'entraînement sont à la base des grands modèles comme GPT et BERT.

Apprentissage Auto-Régressif (Causal LM)

- On modélise la probabilité jointe d'une séquence :

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

- L'entraînement consiste à prédire x_t à chaque pas à partir de $x_{<t}$.
- C'est le paradigme utilisé dans :
  - GPT-1, GPT-2, GPT-3, GPT-4.
  - Les modèles de génération de texte.

Avantage : génération fluide.
Inconvénient : pas de contexte futur, entraînement unidirectionnel.

Apprentissage Auto-Régressif (Causal LM)

- On modélise la probabilité jointe d'une séquence :

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

- L'entraînement consiste à prédire x_t à chaque pas à partir de $x_{<t}$.
- C'est le paradigme utilisé dans :
  - GPT-1, GPT-2, GPT-3, GPT-4.
  - Les modèles de génération de texte.

Avantage : génération fluide.
Inconvénient : pas de contexte futur, entraînement unidirectionnel.

Apprentissage Masqué (Masked LM)

- Paradigme introduit par BERT :

  Remplacer des tokens par un symbole spécial `[MASK]`.

- Le modèle doit prédire les tokens manquants à partir du contexte gauche et droit :

$$P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_T)$$

- En pratique :
  - On masque aléatoirement 15% des tokens.
  - 80% sont remplacés par `[MASK]`, 10% par un autre mot, 10% inchangés.

Avantage : contexte bidirectionnel.
Inconvénient : incohérence entre entraînement (avec `[MASK]`) et inférence (sans `[MASK]`).

Apprentissage Masqué (Masked LM)

- Paradigme introduit par BERT :

  Remplacer des tokens par un symbole spécial `[MASK]`.

- Le modèle doit prédire les tokens manquants à partir du contexte gauche et droit :

$$P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_T)$$

- En pratique :
  - On masque aléatoirement 15% des tokens.
  - 80% sont remplacés par `[MASK]`, 10% par un autre mot, 10% inchangés.

Avantage : contexte bidirectionnel.
Inconvénient : incohérence entre entraînement (avec `[MASK]`) et inférence (sans `[MASK]`).

Comparaison : Causal vs Masked LM

Auto-régressif (Causal LM)

- Unidirectionnel
- Génération naturelle
- Pas de `[MASK]` en entrée

Masked LM

- Bidirectionnel
- Bonne représentation du contexte
- Pas adapté à la génération

Synthèse : *Causal LM* pour la génération, *Masked LM* pour l'encodage profond.

Comparaison : Causal vs Masked LM

Auto-régressif (Causal LM)

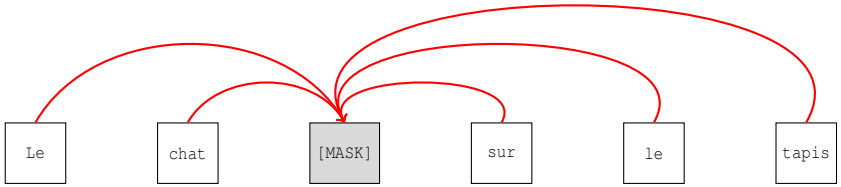
- Unidirectionnel
- Génération naturelle
- Pas de `[MASK]` en entrée

Masked LM

- Bidirectionnel
- Bonne représentation du contexte
- Pas adapté à la génération

Synthèse : *Causal LM* pour la génération, *Masked LM* pour l'encodage profond.

Visualisation : Masquage de tokens



Prédire `dort` à partir du contexte

Optimisation à grande échelle

- Les modèles de type Transformer comptent souvent des milliards de paramètres.
- L'optimisation efficace est donc critique :
  - pour la vitesse de convergence,
  - pour la stabilité de l'entraînement,
  - pour la généralisation du modèle.
- Trois leviers principaux :
  - Algorithmes d'optimisation (ex : Adam)
  - Programmation du taux d'apprentissage (warm-up, decay)
  - Régularisation (Dropout, Weight Decay, Label smoothing)

Un entraînement réussi repose sur la synergie entre ces techniques.

Optimisation à grande échelle

- Les modèles de type Transformer comptent souvent des milliards de paramètres.
- L'optimisation efficace est donc critique :
  - pour la vitesse de convergence,
  - pour la stabilité de l'entraînement,
  - pour la généralisation du modèle.
- Trois leviers principaux :
  - Algorithmes d'optimisation (ex : Adam)
  - Programmation du taux d'apprentissage (warm-up, decay)
  - Régularisation (Dropout, Weight Decay, Label smoothing)

Un entraînement réussi repose sur la synergie entre ces techniques.

Algorithme Adam (Adaptive Moment Estimation)

- Adam est un algorithme d'optimisation stochastique à pas adaptatif. Il combine deux méthodes :
  - **Momentum** : moyenne exponentielle des gradients passés.
  - **RMSProp** : moyenne exponentielle du carré des gradients.
- Soit $\mathcal{L}_t(\theta)$ la fonction de perte à l'étape t. À chaque itération, on calcule :

$$g_t = \nabla_\theta \mathcal{L}_t(\theta_{t-1}) \quad \text{(gradient instantané)}$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(moyenne mobile 1er ordre)}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(moyenne mobile 2nd ordre, au carré)}$$

- **Correction des biais** (car $m_0 = 0$, $v_0 = 0$ induisent un biais vers 0 au début) :

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- **Mise à jour des paramètres** :

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

où :

- η : taux d'apprentissage,
- ε : petit terme de stabilité (typiquement 10^{-8}),
- β_1, β_2 : coefficients de lissage (≈ 0.9 et 0.999 respectivement).

Algorithme Adam (Adaptive Moment Estimation)

- Adam est un algorithme d'optimisation stochastique à pas adaptatif. Il combine deux méthodes :
  - **Momentum** : moyenne exponentielle des gradients passés.
  - **RMSProp** : moyenne exponentielle du carré des gradients.
- Soit $\mathcal{L}_t(\theta)$ la fonction de perte à l'étape t. À chaque itération, on calcule :

$$g_t = \nabla_\theta \mathcal{L}_t(\theta_{t-1}) \quad \text{(gradient instantané)}$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(moyenne mobile 1er ordre)}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(moyenne mobile 2nd ordre, au carré)}$$

- **Correction des biais** (car $m_0 = 0$, $v_0 = 0$ induisent un biais vers 0 au début) :

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t},$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- **Mise à jour des paramètres** :

$$\theta_t = \theta_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

où :

- η : taux d'apprentissage,
- ε : petit terme de stabilité (typiquement 10^{-8}),
- β_1, β_2 : coefficients de lissage (≈ 0.9 et 0.999 respectivement).

Warm-up et décroissance du taux d'apprentissage

- Adam ou tout optimiseur peut être combiné avec un planning du taux d'apprentissage η_t.
- **Problème** : si η est trop grand initialement $\Rightarrow$ explosion des gradients ou divergence.
- **Solution :** stratégie de **Warm-up** :

$$\eta_t = \eta_{\max} \cdot \frac{t}{N_{\text{warm}}}, \quad \text{pour } t \leq N_{\text{warm}}$$

où :

- N_{warm} : nombre d'étapes de chauffe,
- $\eta_{\max}$: taux d'apprentissage maximal.

- **Après le Warm-up : décroissance du taux d'apprentissage** :
  - *Inverse Square Root Decay (Transformer)* :

$$\eta_t = \frac{\eta_{\max}}{\sqrt{t}}, \quad t > N_{\text{warm}}$$

  - *Cosine Annealing* :

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\pi \cdot \frac{t - N_{\text{warm}}}{T - N_{\text{warm}}}\right)\right)$$

  - *Exponential Decay* :

$$\eta_t = \eta_{\max} \cdot \gamma^{-N_{\text{warm}}}, \quad \gamma \in (0, 1)$$

- Ces stratégies assurent stabilité initiale et convergence plus fine.

Warm-up et décroissance du taux d'apprentissage

- Adam ou tout optimiseur peut être combiné avec un planning du taux d'apprentissage η_t.
- **Problème** : si η est trop grand initialement $\Rightarrow$ explosion des gradients ou divergence.
- **Solution :** stratégie de **Warm-up** :

$$\eta_t = \eta_{\text{max}} \cdot \frac{t}{N_{\text{warm}}}, \quad \text{pour } t \leq N_{\text{warm}}$$

où :

  - N_{warm} : nombre d'étapes de chauffe,
  - η_{max} : taux d'apprentissage maximal.

- **Après le Warm-up : décroissance du taux d'apprentissage** :
  - *Inverse Square Root Decay (Transformer)* :

$$\eta_t = \frac{\eta_{\text{max}}}{\sqrt{t}}, \quad t > N_{\text{warm}}$$

  - *Cosine Annealing* :

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}}) \left(1 + \cos\left(\pi \cdot \frac{t - N_{\text{warm}}}{T - N_{\text{warm}}}\right)\right)$$

  - *Exponential Decay* :

$$\eta_t = \eta_{\text{max}} \cdot \gamma^{t - N_{\text{warm}}}, \quad \gamma \in (0, 1)$$

- Ces stratégies assurent stabilité initiale et convergence plus fine.

Régularisation dans les Transformers

- Les modèles Transformer sont complexes $\Rightarrow$ risque de surapprentissage élevé.
- **Dropout** : on applique un masque aléatoire de Bernoulli sur les activations x :

$$y = x \odot \text{Bernoulli}(p), \quad \text{où } p = \text{probabilité de conservation}$$

  - À l'inférence, on multiplie les activations par p pour conserver l'espérance.
- **Weight Decay** : pénalisation L2 des grands poids :

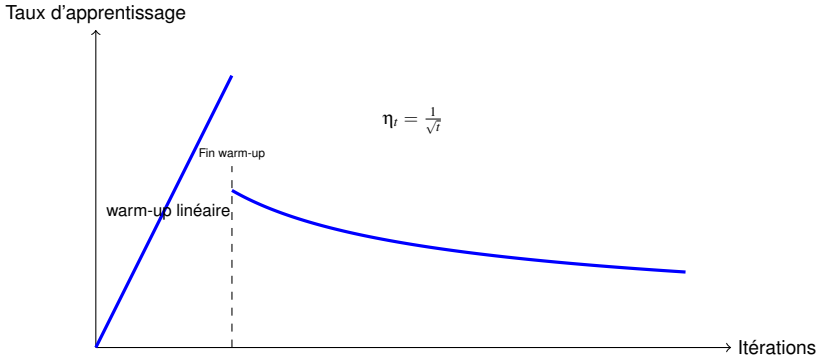
$$\mathcal{L}_{\text{totale}} = \mathcal{L}_{\text{données}} + \lambda \cdot \|\theta\|_2^2$$

  - Encourage les poids à rester petits.
  - Interprétable comme une régularisation bayésienne (prior gaussien sur θ).
- **Label Smoothing** : on évite la surconfiance sur une seule classe :

$$q_{\text{smoothed}}(y) = (1-\varepsilon) \cdot \delta_{y,y^*} + \frac{\varepsilon}{K}$$

  - δ_{y,y^*} : distribution de Dirac (1 si $y = y^*$, 0 sinon).
  - K : nombre de classes.
  - Cela revient à mélanger la vérité avec une distribution uniforme.
- **Effet global** : meilleure généralisation, meilleure robustesse à l'overfitting.

Courbe typique de taux d'apprentissage



Cette courbe illustre une stratégie classique de variation du taux d'apprentissage : une phase de warm-up linéaire où η_t croît progressivement pour stabiliser l'entraînement, suivie d'une décroissance selon une loi en $1/\sqrt{t}$ afin de permettre une convergence plus fine en fin d'entraînement. La transition entre les deux phases se fait au point noté "Fin warm-up".

Alignement des modèles de langage par RLHF

- Les modèles de langage (GPT, T5, etc.) sont préentraînés pour approximer la distribution conditionnelle de texte :

$$f_\theta(y|x) \approx P(y|x)$$

mais cela ne garantit pas que les réponses soient **alignées** avec les attentes humaines (éthique, utilité, sécurité, etc.).

- Objectif de l'**alignement** : modifier la politique f_θ pour qu'elle génère des sorties préférées par les humains.
- Méthodologie classique : **RLHF (Reinforcement Learning from Human Feedback)** :
  1. **Préentraînement** du modèle sur un grand corpus $\mathcal{D}$ (objectif de type maximum de vraisemblance) :

$$\max_\theta \sum_{(x,y)\in\mathcal{D}} \log f_\theta(y|x)$$

  2. **Collecte de préférences humaines** : pour chaque requête x_i, on compare deux réponses y_i^+ (préférée) et y_i^-.
  3. **Entraînement d'un modèle de récompense** $r_\phi(x,y)$ pour approximer les préférences :

$$\max_\phi \sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

avec $\sigma(z) = \frac{1}{1+e^{-z}}$ la fonction sigmoïde.

  4. **Optimisation par renforcement (PPO)** :

$$\max_\theta \mathbb{E}_{y\sim f_\theta(\cdot|x)}\left[r_\phi(x,y)\right] - \beta \cdot \mathrm{KL}(f_\theta \| f_{\text{base}})$$

où f_{base} est la politique préentraînée.

$\mathbb{E}_{y\sim f_\theta(\cdot|x)}[\cdot]$ signifie qu'on calcule l'espérance (la moyenne) sur les différentes réponses y générées par le modèle f_θ conditionnellement à une entrée x.

- Référence clé : *InstructGPT*, Ouyang et al., 2022.

Alignement des modèles de langage par RLHF

- Les modèles de langage (GPT, T5, etc.) sont préentraînés pour approximer la distribution conditionnelle de texte :

$$f_\theta(y|x) \approx P(y|x)$$

  mais cela ne garantit pas que les réponses soient **alignées** avec les attentes humaines (éthique, utilité, sécurité, etc.).
- Objectif de l'**alignement** : modifier la politique f_θ pour qu'elle génère des sorties préférées par les humains.
- Méthodologie classique : **RLHF (Reinforcement Learning from Human Feedback)** :
  1. **Préentraînement** du modèle sur un grand corpus $\mathcal{D}$ (objectif de type maximum de vraisemblance) :

$$\max_\theta \sum_{(x,y)\in\mathcal{D}} \log f_\theta(y|x)$$

  2. **Collecte de préférences humaines** : pour chaque requête x_i, on compare deux réponses y_i^+ (préférée) et y_i^-.
  3. **Entraînement d'un modèle de récompense** $r_\phi(x,y)$ pour approximer les préférences :

$$\max_\phi \sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

     avec $\sigma(z) = \frac{1}{1+e^{-z}}$ la fonction sigmoïde.
  4. **Optimisation par renforcement (PPO)** :

$$\max_\theta \mathbb{E}_{y\sim f_\theta(\cdot|x)}\left[r_\phi(x,y)\right] - \beta \cdot \mathrm{KL}(f_\theta \| f_{\text{base}})$$

     où f_{base} est la politique préentraînée.

  $\mathbb{E}_{y\sim f_\theta(\cdot|x)}[\cdot]$ signifie qu'on calcule l'espérance (la moyenne) sur les différentes réponses y générées par le modèle f_θ conditionnellement à une entrée x.
- Référence clé : *InstructGPT*, Ouyang et al., 2022.

Alignement des modèles de langage par RLHF

- Les modèles de langage (GPT, T5, etc.) sont préentraînés pour approximer la distribution conditionnelle de texte :

$$f_\theta(y|x) \approx P(y|x)$$

  mais cela ne garantit pas que les réponses soient **alignées** avec les attentes humaines (éthique, utilité, sécurité, etc.).
- Objectif de l'**alignement** : modifier la politique f_θ pour qu'elle génère des sorties préférées par les humains.
- Méthodologie classique : **RLHF (Reinforcement Learning from Human Feedback)** :
  1. **Préentraînement** du modèle sur un grand corpus $\mathcal{D}$ (objectif de type maximum de vraisemblance) :

$$\max_\theta \sum_{(x,y)\in\mathcal{D}} \log f_\theta(y|x)$$

  2. **Collecte de préférences humaines** : pour chaque requête x_i, on compare deux réponses y_i^+ (préférée) et y_i^-.
  3. **Entraînement d'un modèle de récompense** $r_\phi(x,y)$ pour approximer les préférences :

$$\max_\phi \sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

     avec $\sigma(z) = \frac{1}{1+e^{-z}}$ la fonction sigmoïde.
  4. **Optimisation par renforcement (PPO)** :

$$\max_\theta \mathbb{E}_{y\sim f_\theta(\cdot|x)}\left[r_\phi(x,y)\right] - \beta \cdot \mathrm{KL}(f_\theta \| f_{\text{base}})$$

     où f_{base} est la politique préentraînée.

  $\mathbb{E}_{y\sim f_\theta(\cdot|x)}[\cdot]$ signifie qu'on calcule l'espérance (la moyenne) sur les différentes réponses y générées par le modèle f_θ conditionnellement à une entrée x.
- Référence clé : *InstructGPT*, Ouyang et al., 2022.

Formulation mathématique du RLHF

- Soit une requête x et deux réponses y^+ (préférée) et y^- (moins bonne).
- On entraîne un **modèle de récompense** $r_\phi(x, y)$ avec une perte par paires :

$$\mathcal{L}_{\text{reward}}(\phi) = -\sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

où σ est la fonction sigmoïde : $\sigma(z) = \frac{1}{1+e^{-z}}$.

- Ensuite, on optimise une politique f_θ pour maximiser la récompense, tout en restant proche de la politique de base f_{base} :

$$\mathcal{L}_{\text{RL}}(\theta) = \mathbb{E}_{y \sim f_\theta(\cdot|x)}[r_\phi(x, y)] - \beta \cdot \text{KL}(f_\theta \| f_{\text{base}})$$

- Cette optimisation est effectuée par **PPO (Proximal Policy Optimization)** :

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right]$$

où :

  - $r_t(\theta) = \frac{f_\theta(y_t|x_t)}{f_{\theta_{\text{old}}}(y_t|x_t)}$ est le rapport de probabilité,
  - $\hat{A}_t$ est l'estimateur de l'avantage (par ex., $\hat{A}_t = r_\phi(x_t, y_t) - b_t$ avec b_t une baseline),
  - ε est un hyperparamètre (souvent 0.1 ou 0.2).

Formulation mathématique du RLHF

- Soit une requête x et deux réponses y^+ (préférée) et y^- (moins bonne).
- On entraîne un **modèle de récompense** $r_\phi(x, y)$ avec une perte par paires :

$$\mathcal{L}_{\text{reward}}(\phi) = -\sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

où σ est la fonction sigmoïde : $\sigma(z) = \frac{1}{1+e^{-z}}$.

- Ensuite, on optimise une politique f_θ pour maximiser la récompense, tout en restant proche de la politique de base f_{base} :

$$\mathcal{L}_{\text{RL}}(\theta) = \mathbb{E}_{y \sim f_\theta(\cdot|x)}[r_\phi(x, y)] - \beta \cdot \text{KL}(f_\theta \| f_{\text{base}})$$

- Cette optimisation est effectuée par **PPO (Proximal Policy Optimization)** :

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right]$$

où :

  - $r_t(\theta) = \frac{f_\theta(y_t|x_t)}{f_{\theta_{\text{old}}}(y_t|x_t)}$ est le rapport de probabilité,
  - $\hat{A}_t$ est l'estimateur de l'avantage (par ex., $\hat{A}_t = r_\phi(x_t, y_t) - b_t$ avec b_t une baseline),
  - ε est un hyperparamètre (souvent 0.1 ou 0.2).

Formulation mathématique du RLHF

- Soit une requête x et deux réponses y^+ (préférée) et y^- (moins bonne).
- On entraîne un **modèle de récompense** $r_\phi(x, y)$ avec une perte par paires :

$$\mathcal{L}_{\text{reward}}(\phi) = -\sum_i \log \sigma\left(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-)\right)$$

où σ est la fonction sigmoïde : $\sigma(z) = \frac{1}{1+e^{-z}}$.

- Ensuite, on optimise une politique f_θ pour maximiser la récompense, tout en restant proche de la politique de base f_{base} :

$$\mathcal{L}_{\text{RL}}(\theta) = \mathbb{E}_{y \sim f_\theta(\cdot|x)}[r_\phi(x, y)] - \beta \cdot \text{KL}(f_\theta \| f_{\text{base}})$$

- Cette optimisation est effectuée par **PPO (Proximal Policy Optimization)** :

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right]$$

où :

  - $r_t(\theta) = \frac{f_\theta(y_t|x_t)}{f_{\theta_{\text{old}}}(y_t|x_t)}$ est le rapport de probabilité,
  - $\hat{A}_t$ est l'estimateur de l'avantage (par ex., $\hat{A}_t = r_\phi(x_t, y_t) - b_t$ avec b_t une baseline),
  - ε est un hyperparamètre (souvent 0.1 ou 0.2).

Architectures Transformer : BERT, GPT, T5

BERT

- **Encodeur uniquement** basé sur l'attention bidirectionnelle.
- Objectif : **Masked Language Modeling (MLM)**.

$$\max_{\theta} \sum_{i \in \text{mask}} \log f_\theta(y_i | x_{\text{masqué}})$$

- Applications : classification, NER, question answering, etc.

GPT

- **Décodeur uniquement**, avec attention causale (unidirectionnelle).
- Objectif : **Next Token Prediction**.

$$\max_{\theta} \sum_{t} \log f_\theta(y_t | y_{<t})$$

- Utilisé pour la génération de texte (autoregressive decoding).

T5 (Text-to-Text Transfer Transformer)

- **Architecture complète encodeur-décodeur**.
- Objectif unifié : tous les problèmes sont formulés comme des tâches de transformation texte $\rightarrow$ texte.

$$x \mapsto y \quad \text{où } x, y \in \mathcal{V}^*$$

- Préentraînement par **Corrupted Span Prediction** :
  - Plusieurs spans consécutifs sont masqués,
  - Le modèle doit reconstruire les morceaux masqués.
- Très flexible pour la traduction, résumé, QA, etc.

Architectures Transformer : BERT, GPT, T5

BERT

- **Encodeur uniquement** basé sur l'attention bidirectionnelle.
- Objectif : **Masked Language Modeling (MLM)**.

$$\max_{\theta} \sum_{i \in \text{mask}} \log f_\theta(y_i | x_{\text{masqué}})$$

- Applications : classification, NER, question answering, etc.

GPT

- **Décodeur uniquement**, avec attention causale (unidirectionnelle).
- Objectif : **Next Token Prediction**.

$$\max_{\theta} \sum_{t} \log f_\theta(y_t | y_{<t})$$

- Utilisé pour la génération de texte (autoregressive decoding).

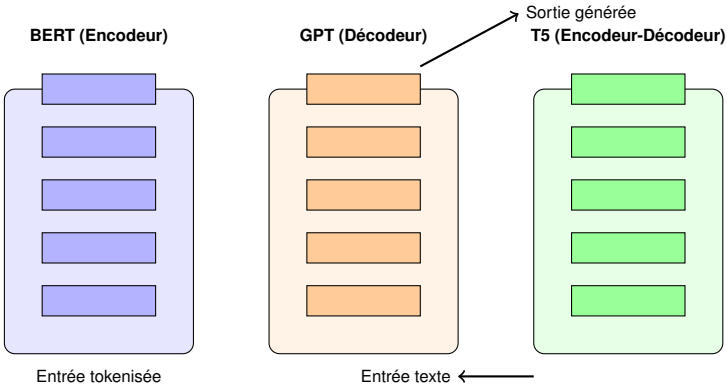
T5 (Text-to-Text Transfer Transformer)

- **Architecture complète encodeur-décodeur**.
- Objectif unifié : tous les problèmes sont formulés comme des tâches de transformation texte $\rightarrow$ texte.

$$x \mapsto y \quad \text{où } x, y \in \mathcal{V}^*$$

- Préentraînement par **Corrupted Span Prediction** :
  - Plusieurs spans consécutifs sont masqués,
  - Le modèle doit reconstruire les morceaux masqués.
- Très flexible pour la traduction, résumé, QA, etc.

Comparaison architecturale des modèles Transformer



Légende

- **BERT** : encodeur bidirectionnel optimisé pour la compréhension (classification, QA).
- **GPT** : décodeur unidirectionnel pour la génération de texte.
- **T5** : architecture encodeur-décodeur unifiée pour toutes les tâches sous forme textetexte.

Conclusion du Module 3

- Modèles de langage modernes combinent :
  - Apprentissage auto-supervisé massif
  - Architectures Transformer spécialisées
  - Optimisation à très grande échelle
  - Alignement via feedback humain (RLHF)
- GPT, BERT, T5 : chacun avec une philosophie propre
- RLHF est aujourd'hui essentiel pour des LLM sûrs et utiles.

Prochaine étape

Module 4 : Préentraînement à grande échelle et fine-tuning spécialisé.

Conclusion du Module 3

- Modèles de langage modernes combinent :
  - Apprentissage auto-supervisé massif
  - Architectures Transformer spécialisées
  - Optimisation à très grande échelle
  - Alignement via feedback humain (RLHF)
- GPT, BERT, T5 : chacun avec une philosophie propre
- RLHF est aujourd'hui essentiel pour des LLM sûrs et utiles.

Prochaine étape

Module 4 : Préentraînement à grande échelle et fine-tuning spécialisé.