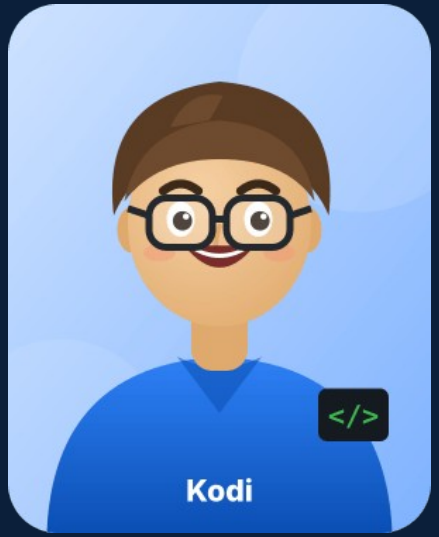
HEF-160 · MISSION 01 / 10

# Modèles génératifs et limites

*Un modèle génératif ne « sait » pas : il prédit ce qui est plausible. Comprendre cette différence est la clé pour s'en servir sans se faire piéger.*







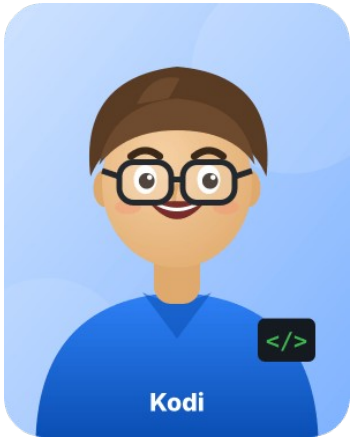
**Badge : Explorateur des modèles**

100 min · avancé

# L'histoire

Un matin, Kodi demande à un assistant d'IA : « Combien de semaines dure le cours de réseaux de HashBox, et qui l'enseigne ? » La réponse arrive instantanément, fluide, sûre d'elle : « huit semaines, enseigné par Mme Lambert ». C'est bien huit semaines. Mais le nom est inventé : aucune Mme Lambert n'enseigne chez HashBox. Kodi n'a rien vu : la phrase était parfaitement écrite.

*« Il n'a pas menti, dit Sam, il a fait exactement ce pour quoi il est construit : produire la suite la plus plausible. Et "Mme Lambert" est un nom plausible. Pour t'en servir en ingénierie, tu dois comprendre comment ça marche et où ça casse. Rien de magique : des probabilités, un contexte limité, et aucune source de vérité interne. »*



# L'histoire (suite)

**Dans ce dernier module, tu apprends à utiliser l'IA comme un multiplicateur de capacité — sans lui déléguer ton jugement. Cette première mission te fait construire un minuscule modèle de langage pour voir, de tes yeux, comment naît une hallucination.**

Ta mission

# À la fin, je serai capable de…

1. expliquer le fonctionnement d'un modèle de langage : jetons, probabilités, prédiction du suivant
2. construire et échantillonner un petit modèle à n-grammes (température, top-k, top-p)
3. mesurer l'effet de la taille du contexte sur la fidélité et sur la créativité
4. expliquer ce qu'est une hallucination et pourquoi elle est fluide et plausible
5. énoncer les limites pratiques : contexte fini, non-déterminisme, connaissances datées, sensibilité à la formulation

Défi d'échauffement

# Défi d'échauffement

Complète : « Le chat boit du _____ ». Plusieurs mots viennent. Lequel est le plus probable ? Le moins ? Maintenant complète : « La capitale de la Mongolie est _____ ». Quelle est la différence entre ces deux situations : dans l'une tu *devines*, dans l'autre tu *sais* (ou tu ne sais pas). Que se passe-t-il si tu devines avec aplomb quand tu ne sais pas ?

Le concept

# Ce qu'est un modèle de langage

Un **modèle de langage** estime la probabilité du **mot suivant** (plus exactement du **jeton** suivant) étant donné ce qui précède :

```
P(jeton suivant | contexte)
```

Pour produire un texte : on **tire** un jeton selon cette distribution, on l'ajoute au contexte, on recommence. C'est tout. Les grands modèles actuels utilisent des **réseaux de neurones** (architecture *transformer*) entraînés sur des quantités énormes de texte, mais le principe reste celui-là.

## Les jetons

Le texte est découpé en **jetons** (*tokens*) : fragments de mots, mots courts, signes. Exemple : « réinitialisation » peut devenir `ré`, `initial`, `isation`. Conséquences pratiques :

Le concept

# Les jetons (suite)

- on **paie** et on est **limité** en jetons, pas en mots (en français, ≈ 1,3 à 1,6 jeton par mot) ;
- les mots rares, noms propres, nombres sont découpés en **beaucoup** de morceaux ;
- les fautes de frappe et le code ont leurs particularités.

Un algorithme classique de découpage : **BPE** (*byte pair encoding*) : on fusionne itérativement les paires de symboles les plus fréquentes.

## De la distribution au texte : l'échantillonnage

| Réglage | Effet |
|---|---|
| **Glouton** (toujours le plus probable) | déterministe, répétitif |
| **Température** `T` | `p_i ∝ exp(logit_i / T)` : `T` bas → plus déterministe ; `T` haut → plus varié et plus risqué |

Le concept

# De la distribution au texte : l'échantillonnage (suite)

| Réglage | Effet |
|---|---|
| **Top-k** | on ne tire que parmi les k jetons les plus probables |
| **Top-p** (noyau) | on garde le plus petit ensemble de jetons dont la probabilité cumulée atteint p |

Même avec les mêmes entrées, un modèle échantillonné **n'est pas déterministe** : c'est une caractéristique, pas un bug. Pour des tâches qui exigent de la reproductibilité, on réduit la température et on fixe la graine quand l'outil le permet, mais cela ne garantit **pas** que la réponse soit exacte.

## Ce que le modèle « sait »

Il a **comprimé des régularités statistiques** de ses données d'entraînement dans ses paramètres. Il n'a :

- **pas de base de faits** interrogeable avec garantie d'exactitude ;

Le concept

# Ce que le modèle « sait » (suite)

- **pas de notion interne** de « je ne sais pas » fiable (il produit quand même quelque chose de plausible) ;
- **aucun accès** à ce qui est postérieur à son entraînement, ni à vos données privées, sauf si on les lui fournit dans le contexte ;
- **aucune vérification** de ses propres sorties (sauf si on lui fait appeler des outils et qu'on vérifie ensuite).

## Les hallucinations

Une **hallucination** est une sortie **plausible mais fausse ou non fondée** : fait inventé, source inexistante, fonction d'une bibliothèque qui n'existe pas, citation fabriquée. Elle est dangereuse parce qu'elle est **fluide** et **assurée** — la forme ne trahit pas le fond.

Le concept

# Les hallucinations (suite)

Pourquoi ? Le modèle optimise la **plausibilité**, pas la **véracité**. Quand le contexte ne contient pas l'information (ou ne la lui rappelle pas), il comble avec ce qui *ressemble* à une réponse.
Facteurs aggravants : question sur un sujet rare, demande de détails précis (noms, nombres, références), contexte ambigu, pression à répondre, température élevée, long texte généré.

## Les limites à connaître

| Limite | Conséquence |
|---|---|
| **Fenêtre de contexte finie** | au-delà, on oublie ou on tronque (mission 3) |
| **Coupure des connaissances** | les faits récents sont inconnus |
| **Sensibilité à la formulation** | deux questions équivalentes, deux réponses différentes (mission 2) |
| **Biais** | reflets des données d'entraînement |

Le concept

# Les limites à connaître (suite)

| Limite | Conséquence |
|---|---|
| **Raisonnement fragile** | arithmétique, logique longue, comptage : erreurs possibles |
| **Confidentialité** | ce qu'on envoie peut être journalisé ou exposé selon le service (mission 9) |
| **Coût et latence** | proportionnels au nombre de jetons |

## Le bon modèle mental

*Un assistant d'IA est un **stagiaire brillant, rapide, très cultivé, qui ne dit jamais « je ne sais pas » et qui peut inventer avec aplomb**. Tu es responsable de ce que tu signes.*

Le concept

# Le bon modèle mental (suite)

Il est excellent pour : brouillons, reformulations, exploration d'options, génération de code **à vérifier**, explication de concepts connus, transformations de format. Il est dangereux pour : faits précis non vérifiés, calculs sans outil, décisions à enjeu sans relecture.

## Capacités et vérification

Plus la tâche est **vérifiable à bas coût** (un test qui passe, une compilation, un calcul qu'on peut refaire), plus l'IA est un excellent multiplicateur. Plus elle est **difficile à vérifier** (analyse juridique, contenu technique pointu), plus il faut un expert humain. C'est le fil rouge de tout le module (missions 7 et 8).

🧠 ***Astuce*** *: quand une réponse d'IA est étonnamment précise sur un détail que tu ne connais pas (un nom, une date, une référence), c'est exactement le moment de la vérifier.*

# Exemple pas à pas

**Problème** : construire un petit modèle de langage à n-grammes entraîné sur des phrases vraies à propos des cours de HashBox, l'échantillonner (température, top-k, top-p), mesurer la part de phrases **fausses mais plausibles** en fonction de la taille du contexte, et illustrer le découpage en jetons par BPE.

```
import math, random, re
from collections import Counter, defaultdict

# ---------------- 1. corpus d'entraînement : uniquement des FAITS VRAIS ---------
   -------
DUREES = {"algorithmes": "dix", "structures": "huit", "réseaux": "huit", "systèmes
   ": "douze", "sécurité": "six", "tests": "cinq"}
PROFS = {"algorithmes": "Ada", "structures": "Bob", "réseaux": "Cléo", "systèmes":
    "Dan", "sécurité": "Eve", "tests": "Flo"}
CORPUS = [f"le cours de {c} dure {DUREES[c]} semaines" for c in DUREES] + [f"le co
   urs de {c} est enseigné par {PROFS[c]}" for c in DUREES]

def est_vrai(phrase):
    m = re.fullmatch(r"le cours de (\w+) dure (\w+) semaines", phrase)
```

Exemple pas à pas

# Exemple pas à pas (suite)

```
    if m: return DUREES.get(m.group(1)) == m.group(2)
    m = re.fullmatch(r"le cours de (\w+) est enseigné par (\w+)", phrase)
    if m: return PROFS.get(m.group(1)) == m.group(2)
    return None
         # phrase mal formée

# ---------------- 2. un modèle à n-grammes ----------------
class ModeleNgrammes:
    def __init__(self, ordre): self.ordre, self.compteurs = ordre, defaultdict(Cou
   nter)
    def entrainer(self, phrases):
        for p in phrases:
            mots = ["<début>"] * (self.ordre - 1) + p.split() + ["<fin>"]
            for i in range(self.ordre - 1, len(mots)):
                self.compteurs[tuple(mots[i - self.ordre + 1:i])][mots[i]] += 1
    def distribution(self, contexte, temperature=1.0, top_k=None, top_p=None):
        c = self.compteurs[tuple(contexte[-(self.ordre - 1):]) if self.ordre > 1 e
   lse ()]
        # logits = log(comptes) ; p ∝ exp(logit / T) = compte^(1/T)
```

# Exemple pas à pas (suite)

```
        poids = {m: n ** (1 / temperature) for m, n in c.items()}
        total = sum(poids.values()); dist = sorted(((m, w / total) for m, w in poi
    ds.items()), key=lambda x: -x[1])
        if top_k: dist = dist[:top_k]
        if top_p:
            cumul, garde = 0.0, []
            for m, p in dist:
                garde.append((m, p)); cumul += p
                if cumul >= top_p: break
            dist = garde
        total = sum(p for _, p in dist); return [(m, p / total) for m, p in dist]
     def generer(self, rng, **options):
        mots = ["<début>"] * (self.ordre - 1)
        while len(mots) < 30:
            dist = self.distribution(mots, **options)
            mot = rng.choices([m for m, _ in dist], [p for _, p in dist])[0]
            if mot == "<fin>": break
            mots.append(mot)
        return " ".join(m for m in mots if m != "<début>")
```

# Exemple pas à pas (suite)

```
# ---------------- 3. fluidité, fidélité et taille du contexte ----------------
def evaluer(ordre, n=400, graine=1, **options):
    m = ModeleNgrammes(ordre); m.entrainer(CORPUS)
    rng = random.Random(graine); phrases = [m.generer(rng, **options) for _ in ran
   ge(n)]
    bien_formees = [p for p in phrases if est_vrai(p) is not None]
    vraies = [p for p in bien_formees if est_vrai(p)]
    copies = [p for p in phrases if p in CORPUS]
    return len(bien_formees) / n, len(vraies) / max(len(bien_formees), 1), len(cop
   ies) / n, len(set(phrases))
print("ordre | phrases bien formées | part de VRAIES parmi elles | copies du corpu
   s | phrases distinctes")
resultats = {}
for ordre in (2, 3, 4, 5):
    bien, vraies, copies, distinctes = evaluer(ordre); resultats[ordre] = (bien, v
   raies, copies, distinctes)
    print(f"  {ordre}   |       {bien:6.1%}        |          {vraies:6.1%}
      |      {copies:6.1%}      |        {distinctes}")
```

# Exemple pas à pas (suite)

```
assert resultats[2][0] > 0.95 and resultats[2][1] < 0.40                    # cont
   exte court : fluent mais souvent FAUX (hallucinations)
assert resultats[5][1] == 1.0 and resultats[5][2] == 1.0                      # co
   ntexte long : fidèle… parce qu'il RECOPIE le corpus
assert resultats[2][3] > resultats[5][3]                                      # et
    la variété (créativité) s'effondre
print("exemples de phrases fluides mais FAUSSES (ordre 2) :")
m2 = ModeleNgrammes(2); m2.entrainer(CORPUS); r = random.Random(7)
fausses = []
while len(fausses) < 3:
    p = m2.generer(r)
    if est_vrai(p) is False and p not in fausses: fausses.append(p)
for p in fausses: print("   •", p)

# ---------------- 4. la température et les autres réglages ----------------
m3 = ModeleNgrammes(2); m3.entrainer(CORPUS)
def entropie(dist): return -sum(p * math.log2(p) for _, p in dist if p > 0)
contexte = ["<début>", "le", "cours", "de"]
print("température | entropie (bits) | probabilité du mot le plus probable après «
```

# Exemple pas à pas (suite)

```
    le cours de »")
for T in (0.2, 0.5, 1.0, 2.0):
    d = m3.distribution(contexte, temperature=T); print(f"    {T:3.1f}     |     {
   entropie(d):5.2f}      |  {d[0][1]:.2f}")
assert entropie(m3.distribution(contexte, temperature=0.2)) <= entropie(m3.distrib
   ution(contexte, temperature=1.0)) <= entropie(m3.distribution(contexte, tempera
   ture=2.0))
# les 6 cours sont équiprobables : on prend un contexte déséquilibré pour voir l'e
   ffet sur le mot le plus probable
m3.entrainer(["le cours de algorithmes dure dix semaines"] * 6)
d_bas, d_haut = m3.distribution(contexte, temperature=0.2), m3.distribution(contex
   te, temperature=2.0)
assert d_bas[0][1] > d_haut[0][1]                                              # T
    basse : l'écart entre probable et improbable s'accentue
top = m3.distribution(contexte, top_k=2); noyau = m3.distribution(contexte, top_p=
   0.5)
print("top-k = 2 →", [(m, round(p, 2)) for m, p in top], "| top-p = 0,5 →", [(m, r
   ound(p, 2)) for m, p in noyau])
assert len(top) == 2 and len(noyau) <= len(m3.distribution(contexte))
```

# Exemple pas à pas (suite)

```
glouton = {m3.generer(random.Random(g), temperature=0.01) for g in range(20)}
print("température quasi nulle : phrases distinctes sur 20 tirages =", len(glouton
   ), "| température 1,0 :", len({m3.generer(random.Random(g)) for g in range(20)}
   ))

# ---------------- 5. BPE : fusionner les paires de symboles les plus fréquentes -
   ---------------
def apprendre_bpe(texte, nb_fusions):
    mots = Counter(texte.split()); vocab = {tuple(m) + ("</w>",): n for m, n in mo
   ts.items()}; fusions = []
    for _ in range(nb_fusions):
        paires = Counter()
        for symboles, n in vocab.items():
            for a, b in zip(symboles, symboles[1:]): paires[(a, b)] += n
        if not paires: break
        (a, b), _ = paires.most_common(1)[0]; fusions.append((a, b)); nouveau = {}
        for symboles, n in vocab.items():
            res, i = [], 0
            while i < len(symboles):
```

# Exemple pas à pas (suite)

```
                if i < len(symboles) - 1 and (symboles[i], symboles[i + 1]) == (a,
    b): res.append(a + b); i += 2
                else: res.append(symboles[i]); i += 1
            nouveau[tuple(res)] = n
        vocab = nouveau
    return fusions
def decouper(mot, fusions):
    symboles = list(mot) + ["</w>"]
    for a, b in fusions:
        i, res = 0, []
        while i < len(symboles):
            if i < len(symboles) - 1 and (symboles[i], symboles[i + 1]) == (a, b):
    res.append(a + b); i += 2
            else: res.append(symboles[i]); i += 1
        symboles = res
    return symboles
TEXTE = "initialisation réinitialisation initialiser réseau réseaux réseau initial
    " * 5 + "algorithme algorithmes algorithmique " * 5
fusions = apprendre_bpe(TEXTE, 30)
```

# Exemple pas à pas (suite)

```
for mot in ("réinitialisation", "réseaux", "algorithmique", "xylophone"):
    print(f"  {mot:18s} → {decouper(mot, fusions)}  ({len(decouper(mot, fusions))}
    jetons)")
assert len(decouper("réinitialisation", fusions)) < len(decouper("xylophone", fusi
   ons))      # un mot rare est découpé en plus de morceaux
```

**Lecture** :

1. Un modèle à **contexte court** (ordre 2) produit des phrases **bien formées** mais majoritairement **fausses** : il mélange les durées et les enseignants entre cours. Ce sont des **hallucinations** : plausibles, fluides, fausses — alors que tout son corpus d'entraînement était **vrai**.
2. Un modèle à **contexte long** (ordre 5) est **toujours fidèle**… parce qu'il **recopie** son corpus : plus de variété. Fidélité et créativité sont en tension. Les vrais modèles évitent le recopiage grâce à leur taille et leur généralisation, mais conservent le risque de combler les trous par du plausible.

# Exemple pas à pas (suite)

3. La **température** modifie la forme de la distribution : basse, elle accentue le mot probable ; haute, elle l'aplatit (plus d'entropie, plus de variété, plus de risque). À température quasi nulle, le résultat devient presque déterministe.
4. **Top-k** et **top-p** tronquent la distribution pour éviter les jetons très improbables.
5. Le **BPE** découpe un mot courant (`réinitialisation`) en moins de jetons qu'un mot rare (`xylophone`) : les mots rares coûtent plus cher en jetons.

# À toi, avec aide

Tu disposes de la distribution suivante pour le jeton suivant, avec ses probabilités brutes :

```
{"semaines": 0.50, "jours": 0.25, "mois": 0.15, "heures": 0.07,
"ans": 0.03}
```

1. Calcule la distribution après application de la température T = 0,5 et T = 2 ($p \propto p^{T-1}$ normalisée).
2. Calcule l'entropie (en bits) de chaque distribution (T = 0,5 ; 1 ; 2).
3. Applique top-k = 2 puis top-p = 0,8 à la distribution d'origine (renormalise).
4. Quelle probabilité a le mot « ans » dans chaque cas ? Dans quel cas risque-t-on le plus de générer un mot improbable ?

À toi, avec aide · réponse

# Indices progressifs

## Indice 1

Élève chaque probabilité à la puissance 1/T, puis renormalise.

## Indice 2

Entropie = $-\Sigma\ p \log_2 p$.

## Indice 3

Top-p garde les mots les plus probables jusqu'à atteindre 0,8 de probabilité cumulée.

À toi, avec aide · réponse

# Réponse

```
import math
P = {"semaines": 0.50, "jours": 0.25, "mois": 0.15, "heures": 0.07, "ans": 0.03}
def temperature(p, T):
    w = {k: v ** (1 / T) for k, v in p.items()}; s = sum(w.values()); return {k: v
    / s for k, v in w.items()}
def entropie(p): return -sum(v * math.log2(v) for v in p.values())
for T in (0.5, 1, 2):
    d = temperature(P, T); print(f"T = {T}: entropie {entropie(d):.2f} bits ; P(an
   s) = {d['ans']:.3f}")
def top_k(p, k):
    garde = dict(sorted(p.items(), key=lambda x: -x[1])[:k]); s = sum(garde.values
   ()); return {a: b / s for a, b in garde.items()}
def top_p(p, seuil):
    cumul, garde = 0, {}
    for m, v in sorted(p.items(), key=lambda x: -x[1]):
        garde[m] = v; cumul += v
        if cumul >= seuil: break
    s = sum(garde.values()); return {a: b / s for a, b in garde.items()}
print("top-k=2 :", {k: round(v, 2) for k, v in top_k(P, 2).items()}, "| top-p=0,8
```

# Réponse (suite)

```
   :", {k: round(v, 2) for k, v in top_p(P, 0.8).items()})
assert entropie(temperature(P, 0.5)) < entropie(P) < entropie(temperature(P, 2))
assert "ans" not in top_k(P, 2) and "ans" not in top_p(P, 0.8) and temperature(P,
   2)["ans"] > P["ans"]
```

À T = 0,5 la distribution se concentre (« ans » devient quasi impossible) ; à T = 2 elle s'aplatit (« ans » passe de 3 % à ≈ 8 %). **Top-k = 2** et **top-p = 0,8** suppriment complètement « heures » et « ans ». Le risque de générer un mot improbable est le plus élevé à **température haute sans troncature**.

# À toi, en solo

**Banc d'essai d'hallucination.** Étends le petit modèle :

1. ajoute à la base de faits une troisième relation (« le cours de X nécessite Y », avec 6 faits vrais) et adapte `est_vrai` ;
2. entraîne des modèles d'ordre 2 à 6 et mesure, pour chaque ordre, trois indicateurs sur 500 phrases : **bien formées**, **fidèles** (vraies parmi les bien formées), **originales** (bien formées, vraies, absentes du corpus) ;
3. ajoute une défense simple : **vérification contre la base** — une phrase n'est conservée que si elle est vérifiée vraie ; mesure le rendement (part conservée) et la création de phrases vraies nouvelles ;
4. montre qu'aucun ordre n'est à la fois très original et très fidèle dans ce petit corpus, et commente en quoi un modèle plus grand diffère ;
5. écris une fonction `verifier(phrase)` qui renvoie `vrai`, `faux` ou `non vérifiable` (et teste-la sur 5 phrases).

À toi, en solo · réponse

# Pistes de correction

```
import random, re
from collections import Counter, defaultdict

DUREES = {"algorithmes": "dix", "structures": "huit", "réseaux": "huit", "systèmes
   ": "douze", "sécurité": "six", "tests": "cinq"}
PROFS = {"algorithmes": "Ada", "structures": "Bob", "réseaux": "Cléo", "systèmes":
    "Dan", "sécurité": "Eve", "tests": "Flo"}
PREREQ = {"algorithmes": "mathématiques", "structures": "algorithmes", "réseaux":
   "systèmes", "systèmes": "algorithmes", "sécurité": "réseaux", "tests": "structu
   res"}
CORPUS = ([f"le cours de {c} dure {DUREES[c]} semaines" for c in DUREES] + [f"le c
   ours de {c} est enseigné par {PROFS[c]}" for c in DUREES]
          + [f"le cours de {c} nécessite {PREREQ[c]}" for c in DUREES])

def verifier(phrase):
    for motif, base in ((r"le cours de (\w+) dure (\w+) semaines", DUREES), (r"le
   cours de (\w+) est enseigné par (\w+)", PROFS), (r"le cours de (\w+) nécessite
   (\w+)", PREREQ)):
        m = re.fullmatch(motif, phrase)
```

À toi, en solo · réponse

# Pistes de correction (suite)

```
        if m: return "vrai" if base.get(m.group(1)) == m.group(2) else "faux"
    return "non vérifiable"

class Modele:
    def __init__(self, ordre): self.ordre, self.c = ordre, defaultdict(Counter)
    def entrainer(self, phrases):
        for p in phrases:
            mots = ["<d>"] * (self.ordre - 1) + p.split() + ["<f>"]
            for i in range(self.ordre - 1, len(mots)): self.c[tuple(mots[i - self.
   ordre + 1:i])][mots[i]] += 1
    def generer(self, rng):
        mots = ["<d>"] * (self.ordre - 1)
        while len(mots) < 30:
            cnt = self.c[tuple(mots[-(self.ordre - 1):])]
            mot = rng.choices(list(cnt), list(cnt.values()))[0]
            if mot == "<f>": break
            mots.append(mot)
        return " ".join(m for m in mots if m != "<d>")
```

À toi, en solo · réponse

# Pistes de correction (suite)

```
print("ordre | bien formées | fidèles | originales (vraies, absentes du corpus) |
   rendement après vérification")
for ordre in (2, 3, 4, 5, 6):
    m = Modele(ordre); m.entrainer(CORPUS); rng = random.Random(ordre)
    phrases = [m.generer(rng) for _ in range(500)]
    bien = [p for p in phrases if verifier(p) != "non vérifiable"]
    vraies = [p for p in bien if verifier(p) == "vrai"]
    originales = [p for p in vraies if p not in CORPUS]
    print(f"  {ordre}   |   {len(bien) / 500:6.1%}    | {len(vraies) / max(len(bie
   n), 1):6.1%} |        {len(originales) / 500:6.1%}                    |  {len(vr
   aies) / 500:6.1%}")
tests = ["le cours de réseaux dure huit semaines", "le cours de réseaux dure six s
   emaines", "le cours de réseaux est enseigné par Cléo", "bonjour le monde", "le
   cours de physique dure dix semaines"]
print([verifier(t) for t in tests])
assert [verifier(t) for t in tests] == ["vrai", "faux", "vrai", "non vérifiable",
   "faux"]
```

# Pistes de correction (suite)

Dans ce petit monde, la **vérification contre une source de vérité** convertit un générateur peu fiable en système fiable : on ne garde que ce qu'on peut confirmer. C'est le schéma de toute l'ingénierie avec l'IA : **générer** (cher en erreurs, peu cher en coût) puis **vérifier** (indépendamment). Un grand modèle généralise bien au-delà d'un corpus de 18 phrases, mais le principe reste : sa sortie est une **proposition**, jamais une preuve.

# Pièges classiques

| Piège | Pourquoi c'est un problème | Comment l'éviter |
|---|---|---|
| **Croire qu'une réponse fluide est vraie** | L'hallucination est plausible et assurée | Vérifier tout détail précis (noms, nombres, références) |
| **Confondre déterminisme et exactitude** | Une température basse ne garantit pas la vérité | Vérification indépendante |
| **Supposer que le modèle connaît vos données ou l'actualité** | Réponses inventées ou périmées | Fournir les sources dans le contexte (mission 4) |
| **Envoyer des données sensibles sans réfléchir** | Fuite de confidentialité | Politique d'usage, redaction (mission 9) |

# Mini-boss

**Fiche d'évaluation d'un modèle jouet.** Construis une « carte de modèle » de ton petit modèle à n-grammes :

1. données d'entraînement (taille, source, couverture) et connaissances absentes ;
2. paramètres de génération et leur effet mesuré (température 0,2 / 1 / 2 ; top-k ; top-p) sur la **fidélité**, la **variété** et la **longueur** ;
3. taux d'hallucination selon l'ordre (2 à 6) et le type de fait (durée, enseignant, prérequis) ;
4. **limites documentées** : contexte, nouveauté, sensibilité à la formulation ;
5. une procédure de **vérification** (source de vérité) et son rendement ;
6. une **recommandation d'usage** : ce pour quoi le modèle est adapté (reformuler, proposer), ce pour quoi il ne l'est pas (affirmer un fait), avec la règle de vérification associée ;
7. 5 lignes pour expliquer à un collègue non technicien pourquoi « il a dit que c'était vrai avec assurance » ne prouve rien.

Mini-boss · réponse

# Pistes de correction

Un bon travail : tableau des taux d'hallucination par ordre et par type de fait (les faits dont la source est éloignée du jeton à prédire — par exemple l'enseignant — sont hallucinés davantage que ceux qui sont adjacents), effet de la température documenté, vérification contre une base et rendement, recommandations d'usage claires. L'explication au collègue insiste sur la distinction plausibilité contre véracité et sur la règle : toute affirmation factuelle d'un modèle est une hypothèse jusqu'à vérification indépendante.

Quiz éclair

# Quiz éclair

**1.** Un modèle de langage produit un texte en :

A. tirant successivement le jeton suivant selon une distribution de probabilités
B. consultant une base de faits garantie exacte
C. copiant des pages web en direct

**2.** Une hallucination est :

A. une erreur de syntaxe
B. une sortie plausible et fluide mais fausse ou non fondée
C. un message de refus

**3.** Une température plus élevée :

A. rend le modèle plus exact

Quiz éclair

# Quiz éclair (suite)

B. supprime les hallucinations

C. rend la distribution plus plate : plus de variété et plus de risque

**4.** 🔁 *Révision HEF-150, mission 10* : Une prévision est bien calibrée quand :

A. les événements annoncés à 70 % se réalisent environ 70 % du temps

B. tous les événements annoncés se réalisent

C. les probabilités annoncées sont toujours de 50 %

**5.** 🔁 *Révision HEF-150, mission 10* : Dans une décision de continuer ou d'arrêter, le coût déjà dépensé :

A. est l'argument principal

B. ne doit pas entrer dans la comparaison des options futures

C. doit être doublé

Quiz éclair · réponse

# Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

# Je réfléchis

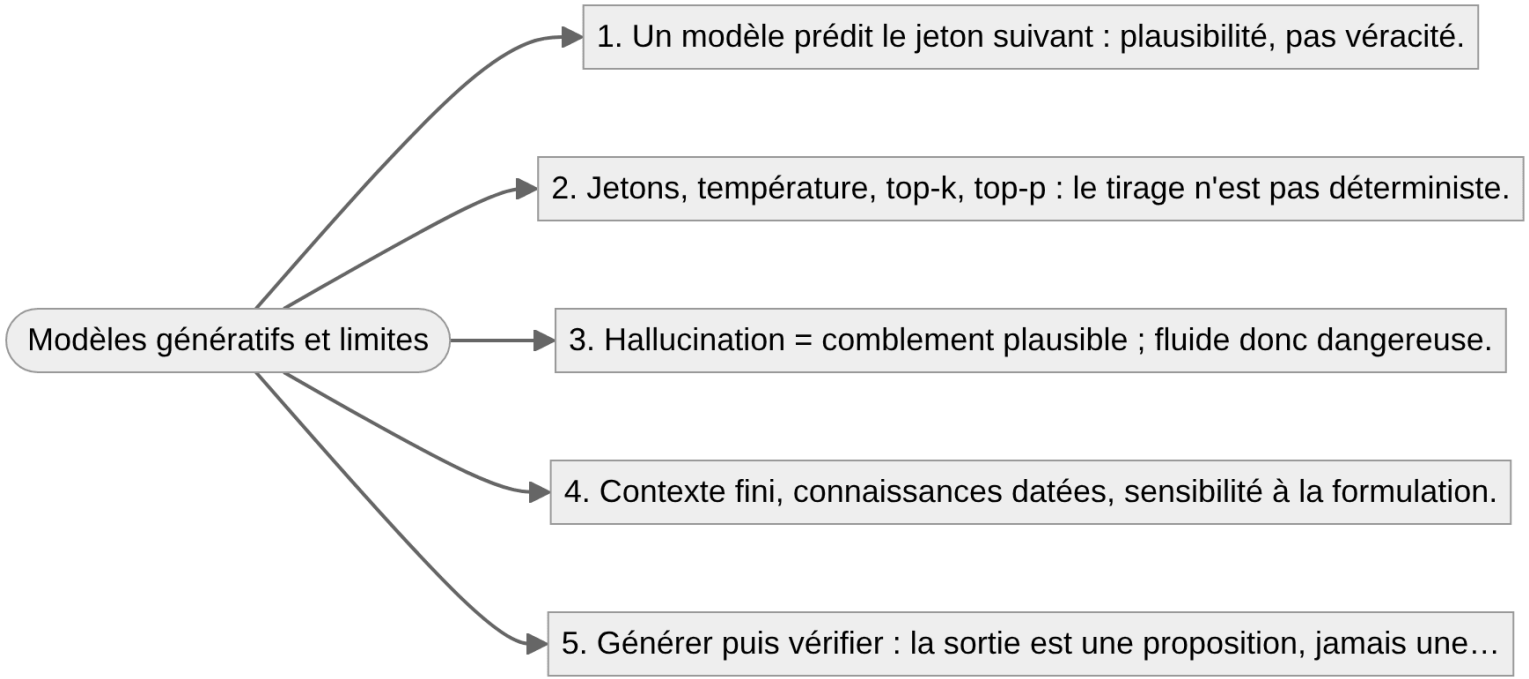
1. Quelle réponse d'IA as-tu déjà acceptée sans la vérifier, et qu'aurais-tu perdu si elle avait été fausse ?
2. Pour quelles tâches de ton travail une erreur de l'IA est-elle bon marché à détecter ?

# À retenir

1. Un modèle prédit le jeton suivant : plausibilité, pas véracité.
2. Jetons, température, top-k, top-p : le tirage n'est pas déterministe.
3. Hallucination = comblement plausible ; fluide donc dangereuse.
4. Contexte fini, connaissances datées, sensibilité à la formulation.
5. Générer puis vérifier : la sortie est une proposition, jamais une preuve.

À retenir

# Le schéma de la mission



MISSION ACCOMPLIE

# +100 XP

## Badge débloqué : Explorateur des modèles

Prochaine mission : Prompt engineering orienté tâche

